NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

BUILDING AN OBJECT MODEL OF A LEGACY SIMULATION

by

Larry R. Larimer

June 1997

Thesis Advisor: Arnold H. Buss

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	**2. REPORT DATE** June 1997	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE BUILDING AN OBJECT MODEL OF A LEGACY SIMULATION	**5. FUNDING NUMBERS**
6. AUTHOR(S) Larimer, Larry R.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army TRADOC Analysis Center Monterey, CA 93943	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release, distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (Maximum 200 words)

The Department of Defense proclamation that all simulations comply with High Level Architecture (HLA) standards prompted the U.S. Army Training and Doctrine Command (TRADOC) Analysis Center (TRAC) to investigate the feasibility of including Janus in future HLA Federations. Janus, one of the Army's most extensively used models, is an stochastic high-resolution simulation. As a procedural legacy model coded prior to the rise of object-oriented programming, there are considerable challenges for Janus to meet HLA requirements.

This thesis proposes a methodology to produce a HLA Simulation Object Model (SOM) for procedurally implemented legacy simulations. The result obtained by using this methodology is a general object model and one or more SOMs. The general object model provides a full object-oriented template of the legacy simulation that is unrestricted by the model's code or the minimum requirement for interoperability. The SOM is derived from the general object model.

This research indicates that procedural legacy simulations can comply with the HLA SOM requirement. In order to achieve this compliance, it is advantageous to first develop the general object model. Additionally, it is important to include an analyst in the SOM development process if federation outputs will be used for analysis.

SOM development facilitated the identification of additional steps necessary to make Janus HLA compliant. This effort will continue with a review of the SOM by Janus code experts and work on a software service that will allow Janus to communicate with other simulations in the HLA specified format.

14. SUBJECT TERMS High Level Architecture, Object Model Template, Simulation Object Model, Federation Object Model, Distributed Interactive Simulation, distributed simulation, object modeling	**15. NUMBER OF PAGES**
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-102

BUILDING AN OBJECT MODEL OF A LEGACY SIMULATION

Larry R. Larimer
Captain, United States Army
B.S., United States Military Academy, 1986

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the
NAVAL POSTGRADUATE SCHOOL
June 1997

Author: ______________________________
Larry R. Larimer

Approved by: ______________________________
Arnold Buss, Thesis Advisor

____
Leroy A. Jackson, Second Reader

____
Richard E. Rosenthal, Chairman
Department of Operations Research

ABSTRACT

The Department of Defense proclamation that all simulations comply with High Level Architecture (HLA) standards prompted the U.S. Army Training and Doctrine Command (TRADOC) Analysis Center (TRAC) to investigate the feasibility of including Janus in future HLA Federations. Janus, one of the Army's most extensively used models, is an stochastic high-resolution simulation. As a procedural legacy model coded prior to the rise of object-oriented programming, there are considerable challenges for Janus to meet HLA requirements.

This thesis proposes a methodology to produce a HLA Simulation Object Model (SOM) for procedurally implemented legacy simulations. The result obtained by using this methodology is a general object model and one or more SOMs. The general object model provides a full object-oriented template of the legacy simulation that is unrestricted by the model's code or the minimum requirement for interoperability. The SOM is derived from the general object model.

This research indicates that procedural legacy simulations can comply with the HLA SOM requirement. In order to achieve this compliance, it is advantageous to first develop the general object model. Additionally, it is important to include an analyst in the SOM development process if federation outputs will be used for analysis.

SOM development facilitated the identification of additional steps necessary to make Janus HLA compliant. This effort will continue with a review of the SOM by Janus code experts and work on a software service that will allow Janus to communicate with other simulations in the HLA specified format.

THESIS DISCLAIMER

TABLE OF CONTENTS

I. INTRODUCTION 1

A. BACKGROUND 2

1. The Janus Combat Simulation 2

a. Janus as an Analysis Tool 3

b. Janus Support Requirements 4

c. Limitations of Janus 4

2. Distributed Interactive Simulation 5

3. High Level Architecture 6

a. Components of HLA 7

b. The Role of the SOM and FOM 8

c. HLA Summary 8

4. Object Oriented Modeling and Design 9

B. STATEMENT OF THESIS 12

II. METHODOLOGY 15

A. CONCEPT 15

B. BUILDING THE GENERAL OBJECT MODEL 16

1. The Object Class Structure 16

2. Attribute/Parameter Table 20

3. Interaction Table 24

4. SOM Lexicon 27

5. Association Table 27

6. Component Structure Table 27

7. Object Model Metadata.. 28

C. BUILDING THE SOM 28

1. The Object Class Structure 28

2. Attribute/Parameter Table ... 29
3. Interaction Table ... 29
4. SOM Lexicon ... 29
5. Association Table ... 30
6. Component Structure Table ... 30
7. Metadata ... 30
D. AFTER SOM TABLE COMPLETION ... 30
1. Review by TRAC-WSMR ... 31
2. Assessment by RTI Developer ... 31
E. ISSUES WITH THE SOM AND ANALYSIS ... 32
1. The Analysis Federate ... 32
2. Including Additional Object Attributes in the SOM ... 33
3. The Multi-SOM Concept ... 35
4. Excluding Interactions from the SOM ... 36
F. CHAPTER SUMMARY ... 36
III. RESULTS ... 39
A. ASSESSMENT OF THE SOM AND SOM DEVELOPMENT PROCESS ... 39
1. Synopsis of General Conceptual Model Methodology ... 39
2. Results of the SOM Development Process ... 41
3. Comparison with other Object Model Development Processes ... 42
B. ADDITIONAL WORK TO BE HLA COMPLIANT ... 45
1. RTI Service Interface ... 46
2. Inclusion in a Federation ... 48
C. CHAPTER SUMMARY ... 48
IV. CONCLUSIONS AND RECOMMENDATIONS ... 51
A. SUMMARY ... 51

B. JANUS COMPLIANCE WITH THE SOM REQUIREMENT 51

C. ADVANTAGES OF THE GENERAL OBJECT MODEL 52

1. Facilitate Model/Federate use in Analysis 52

2. Avoid Omitting Valuable/Critical Aspects of the Model 52

3. Capture Janus Conceptually for Development of Future Models.................. 53

D. INCLUDE AN ANALYST IN SOM DEVELOPMENT.................................. 53

E. DISCUSSION... 54

F. CONCLUSIONS AND FUTURE WORK 54

APPENDIX A. EXAMPLES OF COMPLETED SOM TABLES................................ 57

APPENDIX B. PASCAL CODE FOR ATTRIBUTE/PARAMETER TABLE...............69

APPENDIX C. JANUS OBJECT CLASS HIERARCHY... 75

APPENDIX D. JANUS TARGET ACQUISITION ALGORITHM............................... 79

APPENDIX E. JANUS TUTORIAL 89

LIST OF REFERENCES 113

BIBLIOGRAPHY 115

INITIAL DISTRIBUTION LIST 117

EXECUTIVE SUMMARY

As one of the nation’s largest users of simulation models, the Department of Defense (DoD) is immensely interested in simulation interoperability. Recent advances in technology have demonstrated weaknesses in the current standards for models to communicate and interact. After an intensive period of research, in August 1996 the Defense Modeling and Simulation Office (DMSO) released a standard for simulation interoperability called the High Level Architecture (HLA). The DoD has proclaimed that all simulation models must comply with the HLA standards by fiscal year 2001 or be retired.

The high cost of developing new simulations that comply with the HLA rules for simulation interoperability has prompted many DoD agencies to review the feasibility of modifying currently used models to comply with the HLA standards. The first step toward making a simulation model HLA compliant is to document the model’s capabilities in a document called the Simulation Object Model (SOM).

This thesis proposes a methodology to produce a SOM for a procedurally implemented legacy simulation. The methodology was developed from a theoretical perspective and produces a conceptual mapping of the legacy simulation from a procedural state to an object representation. Using this methodology, object model development is not restricted by the structure and implementation of the simulation code. In addition, the modeler is free to use both operational experience and knowledge of object model methodologies to produce the object model.

The result obtained by using this methodology is a general object model and one or more SOMs. The general object model provides a full object-oriented template of the

legacy simulation without restricting the representation to that subset of the simulation's capabilities required for interoperability. The SOM is derived from the general object model by removing those aspects of the general object model not necessary for simulation interoperability or required for some other purpose by the federation. In this manner, one simulation model may have many SOMs with each SOM targeted at a specific federation or analysis need.

There are potential difficulties incurred by using this methodology. Since the simulation code is not used to derive the model, there may be errors of omission. However, this potential for errors must be weighed against the benefit obtained by the conceptual approach to building the object models. To identify any such errors, the general object model will be reviewed by Janus code experts to verify that the model accurately represents the Janus simulation.

The research and experience resulting from this thesis clearly indicates that procedural legacy simulations, and Janus in particular, can comply with the HLA SOM requirement. In order to achieve this compliance, it is advantageous to first develop a general object model that provides a full representation of the legacy simulation. An analyst can use the general object model to determine if a particular simulation is appropriate to quantify his measures of effectiveness (MOE). Producing the general object model also provides greater confidence that no important aspects of the legacy simulation were overlooked in producing the SOM. Finally, the general object model captures the important qualities of the legacy simulation and can serve as a template for future simulation development.

The HLA guidance gives SOM developers significant flexibility regarding what information will be included in the SOM. Since the analyst must use the model to quantify some specified MOE, an analyst should be included during SOM development if the model will be used in a situation requiring analysis of model outputs. This allows the analyst to ensure that information relevant to capture his MOEs are included in the SOM and will be distributed so that other simulations can also capture that information.

The effort to bring Janus into compliance with the HLA standards will continue. After the object model is reviewed by Janus code experts, tentative plans are in place to begin work on a software service that will allow Janus to communicate with other simulations in the HLA specified format. This project has generated significant interest as a viable means to extend the use of Janus into the HLA environment.

The DoD has invested many resources toward the development and implementation of the HLA. While the concept has proven viable, there is much work remaining before HLA can be considered matured. This thesis has identified some difficulties with modifying a legacy simulation so that it can operate using the HLA. Some larger difficulties such as time synchronization and standard object representation between simulations are yet to be solved.

I. INTRODUCTION

As one of the nation's largest users of simulation models, the Department of Defense (DoD) is immensely interested in simulation interoperability. Budget constraints in recent years have forced the military toward large scale use of simulations and wargames for training, testing, and development of new warfighting equipment and techniques. Advances in technology have allowed several different simulation models to interoperate producing a more complex and realistic representation of the modern battlefield.

For several years, a process called Distributed Interactive Simulation (DIS) has been the standard for simulation interoperability. Recent changes in simulation model applications have demonstrated weaknesses in DIS. Most notably, as the demand for communication between large complex models increased, limitations in network bandwidth and CPU power became apparent in DIS. After an intensive period of research, in August 1996 the Defense Modeling and Simulation Office (DMSO) released a standard for simulation interoperability called the High Level Architecture (HLA). The Department of Defense has proclaimed that all simulation models must comply with the High Level Architecture standards by fiscal year 2001 or be retired.

The high cost of developing new simulations that comply with the HLA rules for simulation interoperability has prompted many DoD agencies to review the feasibility of modifying currently used models to comply with the HLA standards. The first step toward making a simulation model HLA compliant is to document the model's capabilities in a document called the Simulation Object Model (SOM).

This thesis describes the process used to develop a general object model and a HLA SOM for a U.S. Army high resolution, close-combat model called Janus. More importantly, this thesis explores the creation of these models from an analyst's perspective. It describes issues, results, and conclusions directed toward the future use of legacy simulations for analysis within the HLA environment.

A. BACKGROUND

1. The Janus Combat Simulation

Janus is a high resolution, interactive, six-sided, closed, stochastic, ground combat simulation. First developed at Lawrence Livermore National Laboratories to model nuclear effects, the Army's Training and Doctrine Command (TRADOC) Analysis Center (TRAC) at White Sands Missile Range (TRAC-WSMR) accepted responsibility for further development of Janus and produced a modified Janus to meet Army high resolution combat model requirements. Since its fielding in 1978, Janus has been used extensively within the U.S. Army for both training and analysis. Outside the U.S. Army, Janus is used for analysis by RAND Corporation, the United States Marine Corps, and by the armed forces of the United Kingdom, Australia, Canada, France, and Germany. Janus is coded in the procedural FORTRAN 77 programming language and has undergone numerous revisions [1]. Major changes in Janus occurred with version 4.0 which integrated terrain features (roads, water, buildings, vegetation, other man-made features) into the Janus environment. In addition to depicting these features on the Janus graphical terrain, algorithm adjustments in Janus code included consideration for these terrain features in Janus search and detection algorithms and probability of hit and probability of kill calculations. The current version of Janus is version 6.X which adds a multi-sided capability (previously Janus allowed only two opposing forces). Janus 6.X allows up to six sided force interactions [2]. The next step, a "distributed" version of Janus (version 8.0), is currently under development at TRAC-WSMR.

As a high resolution simulation focused on ground combat, Janus models entities down to the individual soldier and vehicle or aircraft level. Up to fifteen homogeneous entities can be aggregated for display and control. Entity interactions with these aggregated units are handled by Janus as if they were individuals. Janus can run at various time scales. It can run in real time as a wargame or man in the loop simulation, or faster than real time for data generation. The limiting factor in the time scale is the number of entities in the scenario. The Janus search and detection algorithm is computationally demanding and slows the run time as the number of entities increases. [1]

a. Janus as an Analysis Tool

Janus has been highly successful as a tool to analyze the effectiveness of new military systems and tactical doctrines. Two components stand out as key elements of Janus' success as an analyst's tool: the database and the post-processor.

The robustness of the database allows the analyst to easily model new military systems and provides the ability to model new and proposed developments to existing systems. It includes nearly every ground vehicle, dismounted system, and Army rotary wing aircraft in U.S. and allied inventories. Most comparable threat nation systems are also portrayed. Systems that are not in the database can be easily created. Over 350 attributes are available in the database for the analyst to model platforms, weapon systems, sensors, projectiles, barriers, and weather. The Attribute/Parameter Table

(Appendix A) provides the reader with a more detailed assessment of the attributes available in the Janus database to model different entities. The most current version of Janus provides representation of limited types of fixed wing aircraft and precision guided munitions. [1]

The Janus Post-Processor provides detailed data on entity interactions that occurred during the execution of a given scenario. These output reports include an artillery fire report, indirect fire ammunition expenditure report, direct fire reports, detection tables, coroner's report, and killer/victim scoreboard. Additionally, Janus provides a supplementary program called the Janus Analyst Workstation. This tool allows the analyst to re-execute scenarios in an "instant replay" format and view events graphically as they occurred. The Janus Analyst Workstation also provides statistical output that is synchronized with the run time of the scenario, providing the analyst with another data analysis tool. [2]

b. Janus Support Requirements

Janus is designed to run on either a VAX computer with a Virtual Memory System (VMS) operating system or a Hewlett-Packard workstation with the UNIX operating system. Under the UNIX operating system, Janus can support up to sixteen different participant workstations. [1]

c. Limitations of Janus

Janus has some limitations for use in analysis. It does not model the area fire effects of direct fire weapons and it does not detail the effects of nuclear weapons beyond the physical effects of a nuclear blast [1]. In its current version, Janus requires man-in-the-loop interaction to change several entity states to include mounting passengers on a carrier entity, dismounting passengers from a carrier entity, changing from non-breaching to breaching state, changing chemical protective over-garment status, and changing hold fire status. While these limitations prevent detailed analysis of some specific systems, Janus is well suited for analysis of most scenarios.

2. Distributed Interactive Simulation

Recent advancements in technology have allowed two or more separate simulations to interoperate using Local Area Network (LAN) or Wide Area Network (WAN) connections. The architecture that has provided simulation interoperability is called Distributed Interactive Simulation (DIS). DIS has made it possible to use distributed simulations for training and analysis. Individual tank simulators may now be linked to high resolution simulations like Janus. The soldiers in the tank simulator fight a battle against Janus entities in a realistic “virtual battlefield environment.” Perhaps most importantly, DIS can link simulations from the different branches of service, providing a vastly improved joint training and analysis capability.

Distributed interactive simulation capability focuses on the standardization of Protocol Data Units (PDUs), packets of information passed from one simulation to the other. Each PDU contains enough information about the entities in one simulation so those entities can be represented in other simulations. The potential to apply DIS to military simulations was realized in 1990. The DIS standards have been revised at semi-annual DIS Workshops designed to improve and extend the protocols. These workshops are modeled after computer and electrical engineering industry standards and includes liaison with the Institute for Electrical and Electronics Engineers (IEEE) [3].

There are limitations to DIS, however. As greater numbers of entities are introduced to distributed simulations, the number of PDU transmissions increase. Current network capabilities can not support the quantity of PDU transmissions necessary to execute distributed simulations with large numbers of entities. Even, as technology improves and network capabilities are able to carry more PDUs, large entity simulations will still be limited by CPU ability to process large quantities of PDUs efficiently. This CPU limitation results in conflicts in time management between simulations. [4] Finally, although DIS provides acceptable interoperability between high resolution simulations with relatively few entities, forward thinking analysts and simulation specialists realized a future need to provide interoperability between high and low resolution simulations. DIS does not support interoperability of high and low resolution simulations. These DIS limitations lead to the development of HLA.[6] Since the release of the DoD HLA documentation in August, 1996, proponents of DIS have begun transition work toward the next generation DIS called DIS++ which will be HLA compliant. [5]

3. High Level Architecture

The Defense Modeling and Simulation Office (DMSO) has the lead role within the Department of Defense for guiding the U.S. Armed Forces toward the most effective use of Distributed Simulation technology. Development of HLA began in 1994 when the DMSO began to look at developing a new architecture for future DoD wide simulations. In January, 1995 the Architecture Management Group (AMG), a working group of simulation experts and computer scientists subordinate to the DMSO, began to develop the baseline definition of HLA. The AMG completed and released the baseline HLA documentation in August 1996. [6] In September 1996 the Department of Defense decreed that all DoD simulations will comply with HLA standardization requirements or be replaced or excluded from Distributed Simulations by the first day of FY 2001 [7].

a. Components of HLA

HLA standardizes the procedures for forming joint interoperating simulations by providing a set of rules for interoperability, a Run Time Interface (RTI) which avoids bandwidth and CPU limitations, and a specified representation of individual simulations and groups of simulations with the Object Model Template (OMT). In HLA, a federate refers to an individual simulation being considered for inclusion in a group simulation. A group of individual simulations becomes a federation when connected through a Run Time Interface and network. The OMT facilitates simulation interoperability by providing object model depictions of the federation called the Simulation Object Model (SOM) and Federation Object Model (FOM). The SOM is an object-oriented model of an individual simulation that is similar to computer science industry object-oriented design standards. The HLA SOM consists of an object class structure table, an object interaction table, an attribute/parameter table, and a FOM/SOM Lexicon (data dictionary). [8] The SOM may optionally include a component structure table, an object associations table, and an object model metadata. The object model metadata is a description of the developmental history of the SOM and execution requirements of the simulation [9].

The FOM includes the same components as the SOM, but is an object model template of the federation created by connecting two or more federates.

b. The Role of the SOM and FOM

Each simulation is described for interoperability using its SOM. The SOM is a “contract” that specifies what the distributed simulation can provide to other simulations and a statement of what it requires in return. Simulations being considered for inclusion in a federation are reviewed for compatibility by comparing their individual SOMs. The result of this comparison is the FOM. The FOM is built from data derived from the individual SOMs, but represents the federation that is formed when the individual simulations interoperate. The FOM is a “contract” between the individual members of the federation that describes what information will be passed between members during the simulation execution.

c. HLA Summary

A typical HLA federation consists of one or more federates. Each federate uses the HLA RTI and the FOM to send (publish) and request (subscribe to) object attribute updates and interactions. Federates subscribe to object attributes and interactions of interest in the FOM and publish those object attribute updates and interactions of interest to other federates. The RTI is used to send and receive the object attribute updates and the interactions through the network. Subscription is thus a filtering process that reduces the network load by interchanging only data of interest during the federation execution. Once the object update is passed to a federate via the RTI the object is appropriately portrayed in the subscribing federate. The object classes in the FOM contain the attributes and interactions to which federates may subscribe and the attributes and interactions that federates may publish. Since HLA uses an object representation for the

SOM and FOM, the object class structure plays an important role in subscription and publication. This topic is revisited in Chapter II.

4. Object-Oriented Modeling and Design

The HLA Object Model Template is based on contemporary object-oriented modeling and design concepts [8]. Object-oriented design is a method of representing entities as distinct units called objects. An object consists of a specified data structure (essentially a record) which contains all of the attributes of the entity, and a set of functions (called methods in object-oriented lexicon) that operate on the object's data. [10] An object can represent a tank, soldier, or sensor, and can represent less well defined entities such as terrain and weather. An object representing a tank can have other objects (a sensor or weapon system) as component parts.

Basic principles of object-oriented design include encapsulation, inheritance, and polymorphism. Encapsulation is the separation of an object's interface and its implementation. One consequence is that an object's attributes should only be changed by the methods that belong specifically to that object. Some of these methods are exposed to allow interactions between objects. Encapsulation prevents corruption of information or object attributes by a mistake in the code or user execution of the program.

Inheritance means that an object can be derived from another (possibly more abstract) object. We can think of an object's hierarchy as a parent-child or lineage relationship. The parent object is defined by its attributes and methods. A descendant of this parent object inherits all the parent's attributes and methods, but will normally define additional attributes and methods. For example, we could specify an object called a "ground vehicle" that contains all the basic attributes that are common to all ground vehicles in the Army. These would include a maximum speed, fuel capacity, crew size, and a sensor. We cannot introduce one of these vehicles to the battlefield because it does not represent a real combat system, and so we refer to it as an "abstract" object. However, we can use it to define child objects that are "concrete." Two descendants of this object are a tracked vehicle and a wheeled vehicle. A tracked vehicle object has all of the attributes of a ground vehicle plus additional attributes that more clearly define it (weapon system, armored protection for crew, smoke generating capability).   Figure 1 depicts this inheritance relationship.

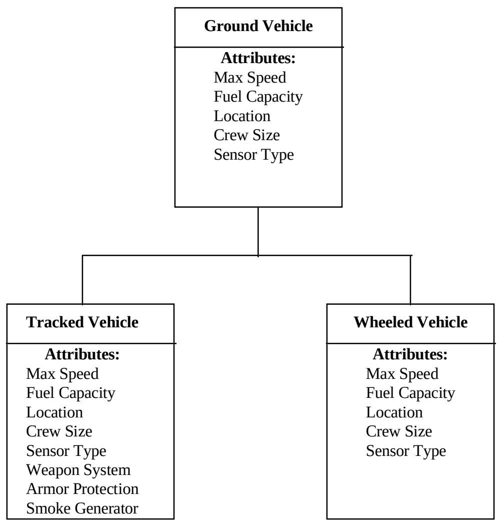


Figure 1. Simple Object Inheritance.

In Figure 1, the tracked vehicle is a descendant of the ground vehicle object and, since it is an object that we could introduce to the battlefield, we refer to it as a concrete object. Inheritance allows us to easily define new concrete objects by selecting the appropriate parent object and adding only selected new attributes or changing attribute values rather than creating a new entity from scratch.

Polymorphism allows descendant objects to use the methods of the parent object, but in a way appropriate for the descendant. For example, the ground vehicle object may have a method called MOVETO that calculates the distance and time to move to a specified new location. Inheritance provides the tracked vehicle and the wheeled vehicle with this method without creating two new MOVETO methods. Polymorphism allows us to modify inherited methods to fit the characteristics of the descendant without changing the name of the method. The MOVETO method inherited by the tracked vehicle would calculate the distance and time to move to a new location, but we could include a routine that also calculates the fuel expended in the move. We now have a modified method that is more appropriate for the descendent object since we are interested in tracking the fuel consumption of the tracked vehicle.

Representation of individual simulations allows an analyst or trainer to efficiently review the OMT component tables to determine what entities are represented in a particular simulation and how the entities are represented. For example, the representation of a tank may be determined by reviewing the OMT attribute/parameter table. While the HLA OMT structure is based on object-oriented models, there are some notable differences between the two. These differences are largely in the design of the OMT and the conceptual assumptions about how the federation will operate. While a pure object-oriented design requires the specification of all the methods associated with each class of object, the HLA OMT does not. Also, the HLA OMT requires an object interaction table, but does not include additional information about state transitions and sequencing of events that object-oriented methodologies require. These differences between the HLA OMT and object-oriented modeling methods represent differences in the way the simulation (program) is portrayed on paper. However, perhaps a more significant difference is the way in which encapsulation and polymorphism are handled in HLA. The conceptual framework of HLA allows the passing of object attributes from one federate to another and allows the receiving federate to change the attribute. This directly violates the encapsulation of an object's attributes. Proponents of HLA argue that this allows a federate with a historically validated (better) algorithm to model specific events or interactions for all the federates resulting in a more accurate federation representation of the real world. [8]

B. STATEMENT OF THESIS

The DoD has made it clear that future military simulations for training and analysis must be HLA compliant. All military organizations that are proponents for individual simulations have been directed by DoD to review their simulations over the next several months to determine if the simulations can be made HLA compliant.

The Janus combat simulation offers to future HLA federations an extensive and successful background of use by many military organizations. Because of this historical success and the U.S. Army's significant investment in Janus, there is a substantial incentive for Janus to participate in future HLA federations. However, Janus, as a legacy model, provides some significant challenges in meeting the HLA requirements. Janus is coded in a procedural language with no well-documented object model definition. This thesis develops a Simulation Object Model of Janus that conforms to HLA standards and facilitates the use of Janus in future Distributed Simulations. More importantly, it will describe the rationale and methodology for creating a Simulation Object Model (SOM) of a high resolution, combined arms combat simulation from an analyst's perspective.

This thesis also outlines precisely which characteristics of Janus have made it so successful for analysis and research. Even if Janus does not survive in its present form, it can pass on a legacy of modeling knowledge and insight to future models.

This thesis consists of four chapters and five appendices. Chapter II describes the methodology used to create the Janus SOM, discusses additional work necessary to verify the SOM representation of Janus, and discusses issues with SOM development and analysis. Chapter III provides a summary of the SOM development methodology, provides an assessment of the SOM product, compares the methodology to another proposed SOM development process, and discusses additional work necessary to make Janus fully HLA compliant. Chapter IV offers conclusions and recommendations directed toward those who may be interested in producing SOMs for legacy simulations and includes a discussion of future work that may occur as a result of this research. Appendix A provides examples of Janus object model tables. Appendix B provides Pascal code necessary to produce presentable hard-copy output in Appendix A. Appendix C displays the Janus object class hierarchy trees. Appendix D describes the Janus target acquisition algorithm. Appendix E is the Janus Tutorial which was developed to facilitate an understanding of the Janus simulation.

II. METHODOLOGY

A. CONCEPT

Modeling Janus as an HLA SOM was approached from a theoretical perspective. The SOM represents a conceptual mapping of Janus from its procedural implementation to an object representation. Unlike other SOMs currently available for review, the Janus SOM is targeted for the Modeling and Simulation Resource Repository (MSRR) rather than a specific federation. The baseline product of this methodology is the general object model. The general object model is a complete representation of the simulation that provides the flexibility to incorporate Janus into any number of future HLA federations.

The general object model provides a means of representing the entire Janus model in a modern, object-oriented form and results in a more complete SOM development process. The set of objects, attributes, and interactions considered was not restricted to those suggested by the HLA SOM. The general object model includes objects and private attributes not required for simulation interoperability, but it is a complete object model representation of Janus from which the SOM can be easily extracted. The object classes in the SOM will be a subset of the classes in the general object model, and the attributes and interactions included in the SOM will be a subset of those in the general object model.

Janus is coded in FORTRAN, which is not an object-oriented language. There are no declarations or syntax that allow easy identification of "objects," let alone specification of the attributes that define these objects. Nevertheless, careful examination of the Janus commands, database, interactions, and algorithms allowed production of an object representation of Janus, including attributes and interactions. Detailed refinement of this working set of objects and interactions resulted in the full general conceptual model.

Building the object model was also approached from an analyst's perspective to facilitate its use in the HLA environment to capture appropriate data (output from a simulation run) to answer quantitative questions. Thus, a key objective was for the Janus object model to capture those qualities of Janus that have made it a successful analysis tool for twenty years.

B. BUILDING THE GENERAL OBJECT MODEL

1. The Object Class Structure

The initial Object Class Structure Table is depicted using an organization chart format. This simple format provided a clearly defined class hierarchy and structure for later documentation in the HLA object model template tables. Two important class hierarchies will be discussed in detail. The first was developed by grouping platforms by physical characteristics to form classes in a parent-child hierarchical structure.

The platform sub-tree of this class hierarchy is based primarily on the Janus database which lists each platform the user might introduce into a scenario. Examples of these platforms include the M1A1 Abrams tank, the M2 Bradley Infantry Fighting Vehicle, and the individual rifleman. These platforms, comprising the concrete platform objects in Janus, were grouped by common attributes and physical characteristics into parent classes. Successive levels of parent classes grouped in the same manner produced a tentative hierarchy of abstract classes culminating at the platform superclass. The resulting object class structure is depicted at Figure 2 with shaded boxes representing concrete classes and unshaded boxes representing abstract classes.

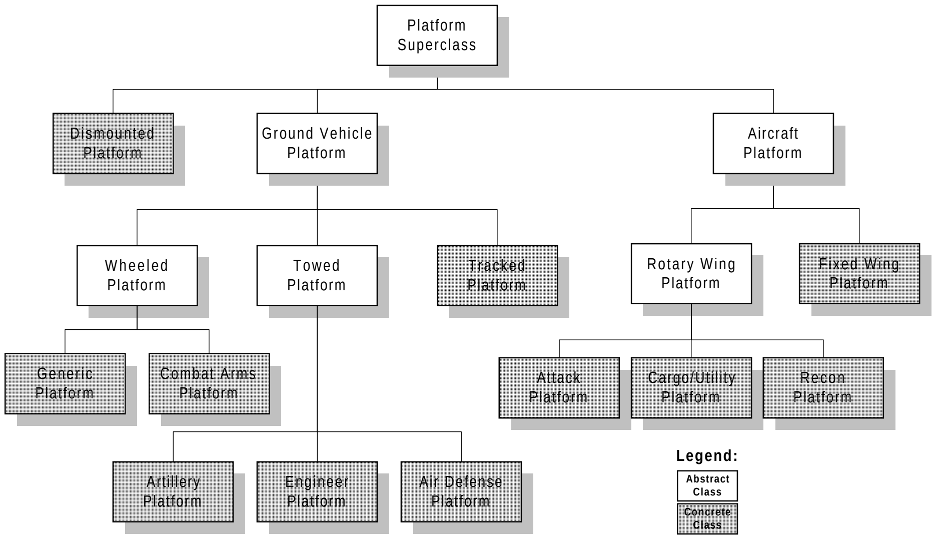


Figure 2. Platform Class Hierarchy.

The second approach to producing a platform class hierarchy used the functionality of each platform as the basis for a parent-child hierarchical structure. Unlike the previous class hierarchy, the concrete objects were first grouped by Battlefield Operating System (BOS) function, and then further sub-grouped by common attributes. The battlefield operating systems distinguish the role of the platform in combat and include: Intelligence, Command and Control, Maneuver, Mobility/Survivability, Air Defense, Fire Support, and Combat Service Support. This is perhaps a more pure object-oriented representation of the Janus simulation and is more suitable for analysis since the analyst is more interested in the functionality of an object than in its physical characteristics. For example, if he is studying the effects of indirect fire, he probably does not care if the artillery piece is towed or tracked. This second platform class hierarchy illustrates the flexibility of the HLA simulation object model to provide more than one appropriate model of a simulation for military analysis and training. The analyst can use this flexibility to produce alternate class hierarchical structures that focus data collection toward quantifying his measures of effectiveness (MOE). See Figure 3 for the BOS platform class hierarchy and Appendix C for other Janus class hierarchies.

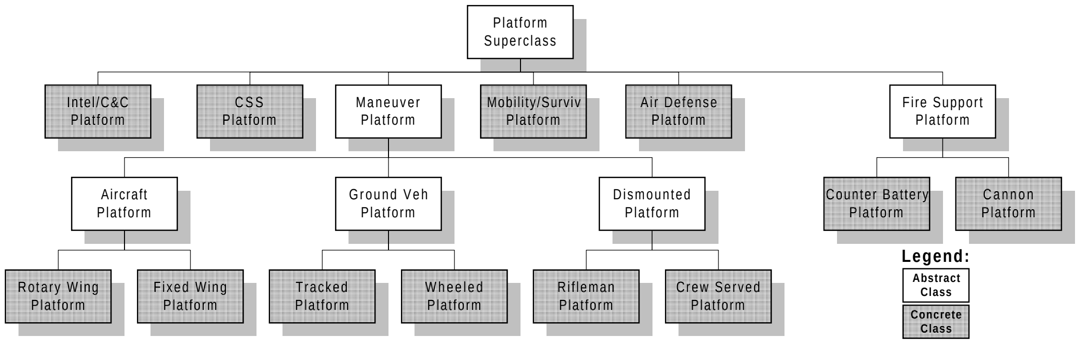


Figure 3. Refined Platform Class Hierarchy.

Although it is not required under the HLA, a terrain class hierarchy was included to produce a more complete Janus object model. In order to represent the terrain in Janus as a class of objects, one must understand how the terrain and the combat platforms associate through the Janus algorithms. From the Janus graphical display, one can discern five basic terrain components: elevation, roads, buildings (towers, etc.), bodies of water, and vegetation. The terrain surface is partitioned into grids with an elevation assigned to each grid cell. The number and size of cells are determined and then fixed within a particular database to support a scenario. The normal cell size is 100 meters by 100 meters. The elevation associates with platform entities in three ways. First, the terrain elevation restricts the speed that platform entities are allowed to move. Rapidly changing elevation reduces an entity's movement speed due to the slower movement rate required for steep terrain. Second, the terrain elevation influences Line of Sight (LOS) calculations in the Janus search and detection algorithm. The search and detection algorithm uses the elevation of each intervening cell between the searching entity and a potentially detected entity to determine LOS. If a terrain cell between the searching entity and the target entity is higher in elevation at any point than a straight line drawn between the entities, then LOS does not exist and no detection can take place. Finally, the terrain elevation object associates with platform entities by providing the platform entities with their appropriate elevation. As a platform entity moves over the terrain, the platform entity periodically updates its location. The longitudinal and latitudinal update comes from the entities previous location, direction of movement and velocity, but the elevation coordinate comes from the terrain. Roads and bodies of water similarly interact with platform entities.

The roads, buildings, vegetation, and water are a separate surface feature layer. The surface feature layer also associates with the battlefield platform entities through the search and detection algorithm. After determining that LOS between two entities is not obstructed by terrain elevation, the algorithm adjusts the probability of detection appropriately based on the type of vegetation or building.

Atmospheric conditions are classified as objects in much the same manner as the terrain. Dust clouds, smoke, and fog all affect the detection algorithm in the same manner as vegetation and buildings. If the detection algorithm identifies one of these objects in the LOS between two objects, it degrades the probability of detection an appropriate amount based on the cloud thickness and type.

The barriers superclass was derived from the Janus initial parameters screen which allows the user to allocate barriers of six different types to each force in a scenario. While these barrier objects are depicted on the Janus graphical display symbolically like the combat platforms, barriers are not found in the Janus database.

Other object classes in the Janus object model are sensor objects, weapon system objects, and ordnance (ammunition) objects. These are components of the platform class and were identified from the Janus database. An example Object Class Structure Table is included at Appendix A.

2. Attribute/Parameter Table

The Attribute/Parameter Table is perhaps the most difficult of the OMT components to construct for a legacy model implemented in a procedural language. In the Janus architecture, every platform entity carries all the attributes available in the platform model. In object-oriented terms, for platform entities, Janus really only has one object class. Every object is an instantiation of this class with only certain attributes filled with values. The values of these attributes define the type of object such as an A-10 fixed-wing aircraft or an individual soldier with a rifle.

Finding all these attributes requires detailed knowledge of the Janus database, commands, graphical display, and to a lesser degree, algorithms. For example, Janus platform entities can be suppressed. An entity that is suppressed has been engaged with direct or indirect fire and is unable to move or respond to the engagement for an amount of time specified by the user during initialization. This response is meant to model a situation where a soldier is receiving enemy fire of such intensity or precision that he is unable to move or return fire. However, there is no attribute in the database or command in the Janus command interface that alerts the user of this capability. One must study the Janus User’s Manual in detail in order to identify this suppression state variable.

Another example is the location attribute. Each entity carries an attribute that tracks its current location (sometimes the last location prior to initiating latest movement). Again, this attribute is not in the database, commands, or graphical display. However, each entity must store its current location to support search and detection calculations and determine engagement outcomes. Tables 1 and 2 depict the basic platform entity attributes identified from the user’s manual, graphic display, and command interface.

User's Manual/Graphical Display
Status(Fully Operational/Casualty)
Suppressed
Entity Number
Smoke (VEES)
Passenger Status
Direction
Location
Destination
Speed

Table 1. Attributes Identified byUser’s Manual/Graphical Display.

Command Interface
Sprint
Hold Fire
Defilade
MOPP
Breach
Helicopter Pop-up

Table 2. Example Attributes Identified by Command Interface.

Every platform entity in Janus, from the individual soldier to the M1 tank, is defined by values entered in these available attributes. After building a class hierarchy and identifying the attributes that define platform objects in Janus, the next step was verify the attributes are correctly distributed at appropriate levels in the class hierarchy. First, those platform class attributes common to all platforms were identified. Examples of these attributes are crew size, location, and maximum movement speed. These became the attributes of the base platform object class (grandfather of all platform class objects). Only those additional attributes necessary to fully describe an object class were included at each descendant level. In other words, only attributes that correspond to the actual object were listed for that class. This process required reexamination of the Janus database to determine which attributes were being used to describe the different platforms, and also required a working knowledge of the military platforms themselves. In some cases, the object model builder must be familiar enough with the platforms modeled to make decisions about appropriate attributes. For example, although Janus will allow any platform to generate engine smoke as a battlefield obscurant (even an aircraft), one familiar with military vehicles knows that only a small number of platform types can generate engine smoke.

Capturing all the attributes that describe the classes of objects and distributing them appropriately through the class hierarchy is only a small part of the detail necessary to complete the attribute/parameter object model table. Each attribute or parameter listed in the table requires 10 other entries. These entries are:

- *Datatype* - integer, real, boolean, etc.
- *Cardinality* - quantity of this attribute/parameter owned by the platform.
- *Units* - units of the attribute (i.e. maxspeed would be KPH).
- *Resolution* - smallest increment in the attribute/parameter value that the simulation can distinguish (Janus cannot distinguish between 25 KPH and 25.01 KPH).
- *Accuracy* - maximum deviation between published attribute/parameter value and intended value.
- *Accuracy Condition* - conditions required for accuracy condition to hold.
- *Update Type* - can the value for the attribute/parameter change.
- *Update Condition* - if the attribute/parameter value can change, what condition causes the change.
- *Transferable/Acceptable* - can the attribute/parameter be transferred to another federate and can this federate receive this attribute/parameter from another federate and do anything useful with it.
- *Updateable/Reflectable* - can the simulation broadcast updates in the attribute/parameter or reflect changes in attributes broadcast by other federates.

These data required for each attribute or parameter are critical to facilitating interoperability between federates. During SOM comparison and FOM development one can identify differences in how the proposed federates represent attributes and resolve conflicts that may otherwise have gone undiscovered. For example, one type of conflict would occur if each proposed federate platform object had an attribute for speed, but the units of one federate was kilometers per hour, while the other federate was in nautical miles per hour. This mismatch of units will be identified when the SOMs are reviewed to develop the FOM. FOM developers could decide what unit of measure will be used by the federation. Simulations that do not use the appropriate units would be required to implement a routine that translates outgoing attribute updates into the proper units for the federation, and another routine to translate incoming attribute updates into the appropriate units for internal use. These routines could be added to the simulation code, or as a service that runs in conjunction with the HLA Run Time Interface (See Chapter III, paragraph B.1. for the discussion of a similar translation architecture).

Since the bulk of the attributes and parameters came from the Janus database, most of the required information for each attribute or parameter could be obtained from the database. For example, values in the database for the parameter optical contrast were represented to four decimal places (i.e., 0.3500), implying a real datatype. Since there is only one optical contrast value in the database for each platform, the cardinality is one. From the target acquisition algorithm, it is noted that the optical contrast is the difference between the optical signature of the object and the background optical signature divided by the background optical signature so the optical contrast is unitless. See Appendix D for a discussion of the Janus target acquisition algorithm. Based on the four decimal places allowed in the database, the resolution of the attribute was determined to be 0.0001 (resolution to four decimal places). Since Janus would publish this attribute simply by pulling the value from the database without manipulation, the accuracy is "perfect" (always publish the exact value that is entered in the database), and the accuracy condition is “always” (there are no special conditions necessary to achieve this accuracy). The optical contrast is a fixed value in the database, so the update type is considered to be “static” (cannot be updated), and the update condition would be “none” (no update condition). The last two attribute/parameter characteristics, Transferable/Acceptable and Updateable/Reflectable, cannot be entered into the object model template with perfect accuracy since the ability of Janus to Transfer/Accept and Update/Reflect attributes is dependent upon the RTI implementation. Entries for these attribute characteristics should be revisited during development of a RTI service.

3. Interaction Table

There are three places in the Janus simulation to identify interactions between objects. The first source is the graphical interface. During a Janus simulation run, many interactions are portrayed graphically on the Janus battlefield screen. Engagements are the primary interactions depicted between platform entities. Between barriers and platform entities, interactions result in the destruction of the platform entity or the halting of the platform entity’s movement.

The second source for interactions is the Janus command interface. The user can direct a platform entity to mount onto another platform entity or to dismount from another platform entity. Also, a platform entity that is capable can be commanded to breach a barrier object, or generate a smoke cloud. The user can command artillery and mortar platforms to execute indirect fire missions with the potential to suppress or destroy platform entities, or to generate smoke clouds. Finally, the user can direct a resupply platform to transload supplies to a combat platform. The Janus Tutorial (Appendix E) produced to facilitate research for this thesis provides the reader with a basic understanding of the Janus command interface.

The third source for interactions is the Janus user's manual. An example of an interaction discovered in the user's manual is the production of a dust cloud by an indirect fire object. The user's manual reveals that artillery rounds impacting with soil creates a dust cloud even if the rounds are not specifically designed to generate smoke. Table 3 depicts the primary interactions identified in Janus.

Platform vs. Platform
Engage with Direct Fire (Graphics Screen)
Mount on/Dismount from Platform (Command Interface)
Resupply Combat Platform (Command Interface)
Radar/Aircraft Acquisition (User's Manual)
Platform vs. Barrier
Encounter Obstacle (Graphics Screen)
Encounter Minefield (Graphics Screen)
Breach Minefield (Command Interface)
Platform vs. Atmosphere
Generate VEES Smoke Cloud (Command Interface)
Generate Grenade Smoke Cloud (Command Interface)
Generate Large Area Smoke Cloud (Command Interface)
Indirect Fire Ordnance vs. Atmosphere
Generate Smoke Cloud (User's Manual)
Generate Chemical Cloud (User's Manual)
Indirect Fire Ordnance vs. Barrier
Emplace FASCAM Minefield (User's Manual)
Indirect Fire Ordnance vs. Area
Casualty Generating Impact (Graphics Screen/User's Man.)
Barrier vs. Atmosphere
Generate Smoke Cloud (Algorithm/Graphics Screen)
Atmosphere vs. Platform
Chemical Cloud Contamination (User's Manual)

Table 3. Interactions.

After all the interactions portrayed by Janus had been identified, the next step was documenting the initiating and receiving classes. Some of these associations are fixed and easily portrayed. For example in every case, a smoke pot object interacts with the atmosphere by changing the attributes of a smoke cloud object (giving it volume, density, etc.). In other cases, the association is less well defined. The direct fire engagement interaction between platform objects can be turned on or off by the Janus user. An M1

Abrams tank modeled by Janus must be specifically authorized to engage other objects in the database by platform type (BMP, BTR-60, T-80 Tank). Therefore, this interaction reflected in the interaction table portrays engagement interactions that Janus is capable of initiating, sensing, or reacting to rather than what may be allowed during any specific Janus scenario run.

In some cases the attributes associated with the initiating and receiving object classes were identified by inspection (i.e., Alive/Dead Status for the receiving platform in a direct fire engagement). Other attributes were more difficult to identify and required inspection of the algorithms and user’s manual. During this phase, several attributes were identified that were previously overlooked. An example is the mount platform interaction. The Janus user’s manual specifies that a passenger entity cannot engage other entities (the mounted entity is in hold fire status). Additionally, an entity with another entity mounted on it can be identified visually on the battlefield graphics screen. This implies that the carrier entity knows that an object is mounted on it, and therefore must have an attribute (“passenger status”) which is a binary variable indicating whether or not a passenger is mounted. Other examples of attributes identified during completion of the interaction tables are ammunition quantity on hand and fuel quantity on hand.

Finally, interaction parameters are those attributes and parameters necessary for the appropriate algorithm to process the interaction and adjust the attributes of the initiating and receiving objects. Again, several attributes and interaction parameters were identified that had previously been overlooked.

4. SOM Lexicon

The OMT Lexicon required research into the definitions of object parameters and attributes. Most of this information was found in the Janus Database Manager's Manual, Janus User's Manual, or by inspection. Some cases such as the transmission factor attribute in the atmosphere class of objects required a detailed inspection of the target acquisition algorithm.

5. Associations Table

There are two primary associations in Janus: the terrain/platform association and the weapon system/ordnance association.

All platform entities are associated with the terrain elevation object. As a platform moves across the terrain, the platform updates the elevation component of its location through this association. This association is important for the Janus object model, but not for the Janus SOM since the terrain object will not typically be included in the SOM.

Ordnance objects are associated with weapon system objects. The weapon system object fires or launches the ordnance object. One could argue that this relationship is an interaction rather than an association, being a momentary relationship rather than a lasting relationship. However, the relationship between ordnance and the weapon system is a usage relationship characterized by the HLA OMT Extensions Reference, Version 1.0, as an association rather than an interaction.

6. Component Structure Table

The component structure table followed directly from the object class structure table. The Janus database clearly specifies that each platform has three sensor objects and zero to fifteen weapon system objects. If the platform contains a weapon system object, it necessarily must have appropriate ordnance objects. Ordnance objects were considered a component of the platform rather than the weapon system because the quantity of ordnance is more closely associated with the platform than with the weapon system. Different platforms may carry the same weapon system but different types or quantities of ordnance objects. The Janus database allows each platform entity to carry any number of ordnance objects (limited only by user discretion).

7. Object Model Metadata

The object model metadata table is simply a history of the simulation, documentation of the SOM development, and proponent for the SOM. This table was completed for the general model so that it could be incorporated in the SOM without change.

C. BUILDING THE SOM

Producing an HLA SOM from the general object model of Janus consists primarily of removing those items not necessary for the SOM. Since the SOM is meant only to facilitate interoperability between individual simulations, to comply with the HLA those pieces of the general object model that will not play a part in inter-simulation operation are excluded from the SOM. This section describes the reasoning used to pare down the general object model to arrive at an HLA compliant SOM.

1. The Object Class Structure

The class hierarchy table requires very little refinement to produce the SOM from the general object model. Since the terrain representation is typically internally represented in all federates, the terrain base object and all descendent objects are removed from the general object model. All other objects are important to producing an accurate representation of Janus through HLA and are included to produce the class structure table for the SOM.

2. Attribute/Parameter Table

The Attribute/Parameter Table is the one most subject to modification to produce the SOM. Since the parameters support interactions, all parameters in the table are included in the SOM. There are two categories of attributes that must be included in the SOM: 1) attributes that support interactions, and 2) attributes necessary to represent the functionality or current state of the object they define in another federate.

Those attributes that were not in the above two categories were removed from the general object model. Removed attributes include: maxspeed, weapon range, chemical transmit factor, swim type, fuel tank size, and many others. The eliminated attributes share the following characteristics; 1) they all contain fixed values with no potential for dynamic change, 2) they are not necessary to resolve potential interactions, 3) they are not necessary to portray the Janus objects functionality in another federate.

3. Interaction Table

All interactions available in Janus have the potential to occur between federates in a federation run and therefore are included in the SOM.

4. SOM Lexicon

The SOM Lexicon simply defines all those objects, attributes, and parameters retained from the general object model to form the SOM.

5. Association Table

The terrain object/platform entity object association is eliminated from the Association Table since the terrain object is not retained in the SOM. See paragraph II.C.1 for explanation of terrain object exclusion from the SOM. Other associations would be eliminated if they did not support interoperability. This would occur if an association existed between objects in Janus, but no potential existed for the same association to occur between an object in Janus and an object in another federate. There are none in this case.

6. Component Structure Table

Similar to the Association Table, component relationships that do not support interoperability are removed. In this case, all component relationships were included in the SOM.

7. Metadata

Entries in the Object Model Metadata change only if they refer specifically to the SOM. All information describing the simulation and its history remains unchanged.

D. AFTER SOM TABLE COMPLETION

While significant work has gone into the development of the Janus SOM, it is not yet a finished product. The general object model and the SOM are still only approximate models of the Janus simulation. Two additional reviews are required to refine the object models and assess the feasibility of RTI implementation. The first is a detailed review of the general object model and SOM by competent Janus experts to confirm that they are a valid representation of the Janus simulation. The second is a review by a potential RTI implementer to verify that it is feasible to implement an interface to the RTI. This interface must extract appropriate data from Janus and use the RTI services to publish that data during federation execution as portrayed in the SOM. The interface must process information to which the Janus federate subscribes and use this data in Janus in a meaningful way. These reviews will likely identify deficiencies in the general object model and SOM that will require correction and these reviews will yield the legitimate object models. This section describes a proposed SOM review process consisting of a quality review and an implementation review.

1. Review by TRAC-WSMR

As the proponent agency for the Janus simulation, TRAC-WSMR employs analysts, senior programmers, and managers with thousands of hours of experience using Janus as an analysis tool and with reviewing, refining, and improving Janus code. TRAC-MTRY is coordinating to arrange a TRAC-WSMR review of both the general object model and the SOM. The TRAC-WSMR review will ensure that the models are an accurate representation of the Janus simulation. Additionally, the TRAC-WSMR review will assess how well Janus as a federate can interoperate with other likely federates.

2. Assessment by RTI Developer

The RTI implementer will review only the SOM. This review will ensure that the information required to support the SOM representation can be extracted from Janus and provided through the RTI. Since Janus is not a distributed model, the RTI interface must monitor dynamic attribute updates and interactions in Janus, extract this information in the appropriate format, and publish it using the RTI services. When receiving subscribed data from other federates, the RTI interface must transform these updates into a form understood by the Janus simulation. This review will require detailed knowledge of the Janus simulation code and HLA RTI software. It will require a significant effort to implement the RTI interface. While this implementation is required to validate the implementation feasibility and measure the performance characteristics for the SOM, this review can effectively estimate the feasibility of implementation.

E. ISSUES WITH THE SOM AND ANALYSIS

One of the potential benefits of the HLA as described in the introduction is to provide the analyst with distributed simulation capabilities that include increased numbers of entities without bandwidth and CPU limitations. Additionally, with the HLA an analyst could form a federation for a specific study using the most widely accepted models for every separate system portrayed in the federation. In other words, if the analyst is studying Patriot air defense missile system performance against the most modern fighter aircraft, he could use the simulation that best portrays the Patriot system as one federate and the simulation that best portrays modern aircraft as another federate. In theory, this federation would result in a better model than one developed from scratch. This section provides a look at some of the issues that must be considered when producing an HLA SOM that may impact on the federate or federation's use as an analysis tool.

1. The Analysis Federate

The analysis federate has been proposed as a method of capturing data across a federation during the course of a federation run [12]. The analysis federate subscribes to those object classes whose updates and interactions will allow quantification of selected measures of performance. It simply records object updates and interactions as they occur.

Thus, the analysis federate will operate very much like the Janus post-processor. However, it will be much more flexible that the Janus post-processor because it can subscribe to any information published by the federation. For example, if the analyst is only interested in Air Defense weapon performance, he need only subscribe to Air Defense and Aircraft platform updates. On the other hand, if the analyst is interested in resupply operations (not provided by the Janus post-processor), he can subscribe to Combat Service Support platform updates. Additionally, the analyst is not required to co-locate with one of the federates to gather data. The analysis federate need only be connected through a RTI to the federation network.

The alternative to the analysis federate is to capture data to quantify measures of effectiveness at the individual federate level and consolidate the data by some other means. While data consolidation may not always be necessary, there could be difficulties with the consolidation process should it be necessary.

The analysis federate has great potential to let the analyst take advantage of the power and flexibility of the HLA. However, if the specific data required are not published, the analysis federate is useless. This leads to a discussion of the value of including or excluding additional pieces of the general object model in the SOM to support the federate’s use as an analysis tool.

2. Including Additional Object Attributes in the SOM

The analysis federate proposal and similar passive data collection federate proposals make the case for additional information in the SOM beyond that necessary to facilitate interoperability among federates. In order to conduct a meaningful analysis of the data produced by a federation run or series of runs, additional information may be needed in the SOM.

An example of this is detection data. In a study to measure force performance as a function of sensor capabilities, it may be important to capture data such as what time entity 01 on side 1 detected entity 101 on side 2, the locations of the entities when the detection occurred, the type of sensor used in the detection, and other factors associated with the detection. Since the relationship between the detecting and detected entities is not an interaction, association, or component relationship, the parameters and attributes that support this event are not included in the SOM under the interaction, association, or component tables. The detecting entity has an attribute called target_list that consists of an array filed with the enemy force entities that are currently acquired. This list is updated when a detection occurs. However, because the detection event does not affect federation interoperability, it would not be included in the attribute/parameter table of the SOM or FOM, and therefore, would not be published. This detection data may be captured in one or more federates for the entities belonging to that federate only. One way to capture detection data across an entire federation run is to publish the target_list updates for each federate and capture this data in the analysis federate.

To facilitate the collection of detection data, the target_list attribute could be included in the SOM. In the same way many other attributes that do not affect interoperability may be included in the SOM to facilitate data collection and consolidation. Unfortunately, a single SOM that attempts to include all possible attributes that may be necessary for future data collection requirements could become to unwieldy and inefficient. Publishing all the attribute updates in the general object model could result in similar bandwidth and CPU limitations that occur in DIS.

One possible solution to the data collection/attribute inclusion dilemma is to develop more than one SOM for a single prospective federate. This concept will be discussed in more detail in the next paragraph.

3. The Multi-SOM Concept

The general object model provides a comprehensive view of the Janus simulation. Since the federation's FOM draws from the individual federate SOMs, one could consider using the general object model as a universal SOM for Janus. However, there are two problems with this approach. First, the federation developers would have to wade through the general object model to identify only the appropriate objects, interactions, attributes, and parameters necessary for the particular federation. Second, while the general object model includes numerous attributes that could potentially be useful in quantifying different measures of effectiveness, it also includes many attributes that will probably never be necessary for these types of studies and clearly are not needed to facilitate interoperability. Different SOMs can be derived from the general object model to support a variety of analysis requirements. A sensor analysis specific SOM could be designed to publish that information necessary to interoperate and include detection data and sensor attributes. A weapon system analysis SOM could provide additional publishable attributes associated with weapon system performance. This multi-SOM concept would be efficient. Attributes unnecessary to a particular project and to federate interoperability are not included in the project-specific SOM. The result is a streamlined federate selection and FOM development process.

4. Excluding Interactions from the SOM

If objects from other federates are expected to interact with the Janus federate's objects, these interactions must be included in the SOM. An interaction could be excluded from the SOM if the interaction will not play a role in the federation interoperation. This would occur if objects in Janus had no potential to perform a specific interaction with objects in other federates.

However, like the argument to include attributes not necessary to interoperability, it could be important to include these interactions in the SOM to support data gathering by publishing the interaction data whether or not the interactions occur within the Janus federate or between the Janus federate and another federate.

F. CHAPTER SUMMARY

The goal of this chapter was to provide the reader with a thorough understanding of the General Conceptual Object Model methodology for producing a HLA SOM. The methodology produces an object model of a legacy simulation that is not restricted by the implementation of the procedural legacy code. Since the object model was produced from a conceptual approach, it is necessary for Janus code experts to review the model to ensure that it accurately represents the Janus simulation. This review will identify possible errors of omission, as discussed in the following chapter. Additionally, since Janus was not developed to be a distributed simulation, a review by a potential RTI service implementer is necessary to ensure that it is feasible for Janus to publish and subscribe to that data indicated in the SOM.

Before a fully HLA capable Janus can be used as an analysis tool, there are issues to address concerning the HLA and analysis. Proposals such as the analysis federate and

Multi-SOM concept, and concerns about what aspects of a simulation are included in the SOM should be clarified before HLA federations are used for analysis.

III. RESULTS

A. ASSESSMENT OF THE SOM AND SOM DEVELOPMENT PROCESS

This section reviews the methodology used to develop the SOM, discusses the results of the SOM development process, and compares the methodology to a SOM development process proposed by Lutz [11].

1. Synopsis of General Conceptual Model Methodology

The methodology used to create the Janus SOM can be viewed in general as a two-step process. First, the general object model is built from the simulation. Second, the SOM is extracted from the general object model.

Development of the general object model requires a working knowledge of the simulation being modeled. In this case, the user's manual, graphic user interface, database, database manual, and software design manual provided sufficient knowledge of the simulation to produce the general object model.

The basic steps in the general object model development were (also see Figure 4):

- Identify all the instantiable objects in the simulation.
- Identify all the attributes used to define the instantiable objects.
- Develop an object class structure.
- Assign the attributes to the appropriate class level for each object class.
- Identify all possible interactions and associated interaction parameters. (This procedure also identifies some additional object attributes previously overlooked)
- Review the total set of object attributes to include those attributes identified while working on interaction data and verify that new attributes do not reveal additional attributes that were previously overlooked.
- Review the interaction table and verify all necessary attributes and parameters are noted.
- Use above information to complete object model tables.

It must be noted that the above steps produced the bulk of the information necessary to complete the object model tables. Not included above are development of the component structure table, association table, object model lexicon, and the additional information necessary to define attributes/parameters (datatype, units, accuracy, etc.). The component structure and associations were identified after the process outlined above was complete and before work on the tables began. The object model lexicon and attribute parameter definitions were entered as the tables were produced using the Aegis Research Object Model Development Tool (OMDT) software [16]. The OMDT allows the user to enter object model lexicon and attribute/parameter definitions from the object class structure table. In this way, the OMDT greatly simplifies the task of entering this information and avoids the difficulty of switching frequently from attribute/parameter table to object model lexicon table and back to make appropriate entries.

After the general object model was completed, it was a relatively simple matter to

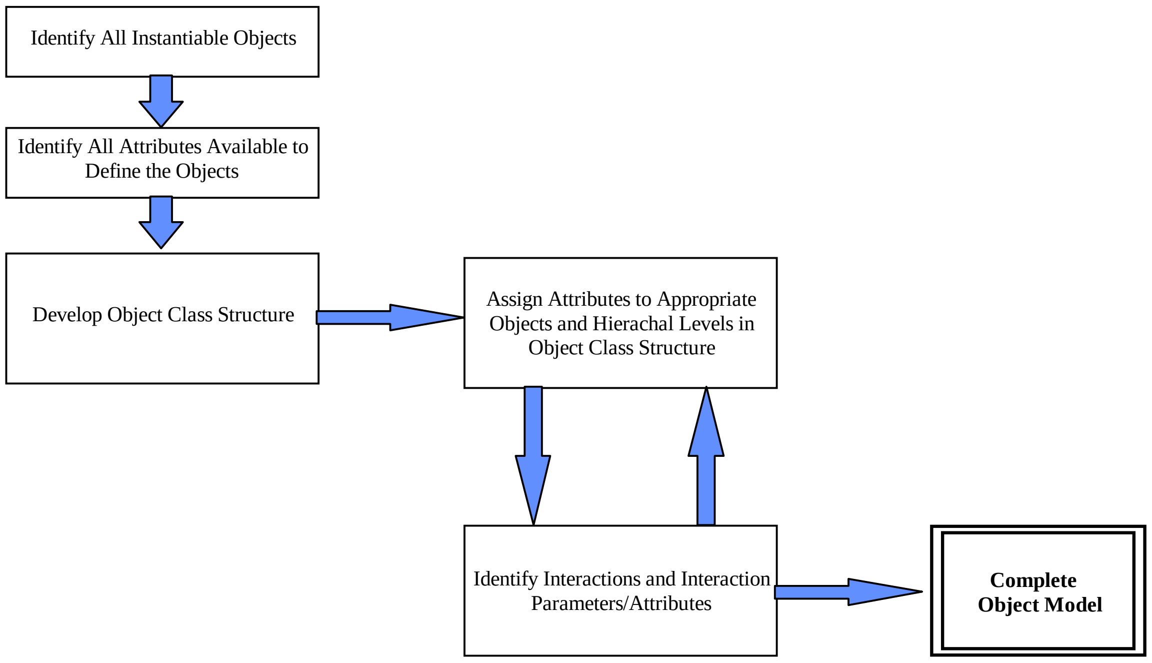

Identify All Instantiable Objects
Identify All Attributes Available to Define the Objects
Develop Object Class Structure
Assign Attributes to Appropriate Objects and Hierachal Levels in Object Class Structure
Identify Interactions and Interaction Parameters/Attributes
Complete Object Model

Figure 4. Graphic Representation of Object model Development Methodology.

extract the appropriate table entries to create the HLA SOM. As discussed earlier, there may be compelling reasons to include additional information in the SOM above the minimal requirements for simulation interoperability. Additionally, should it be deemed necessary or appropriate, multiple SOMs could easily be extracted from the general object model each targeted at a specific HLA federation.

2. Results of the SOM Development Process

Despite the many hours and thought that have gone into the SOM development process outlined in this thesis, there may be errors or omissions in the completed Janus general object model and SOM due to the complexity of the Janus simulation and the approach chosen. This section is a self assessment of the accuracy of the methodology and the quality of the finished general object model and SOM prior to review by Janus model experts.

Errors of omission are most likely to occur in the attribute/parameter table and the interaction table since Janus uses hundreds of attributes to define the many classes of objects. As pointed out in the Chapter II, an overwhelming percentage of these attributes were easily identified by reviewing the database, graphic user interface, user’s manual, and software design manual. However, some attributes could only be identified by inference. The existence of this small set of attributes indicates there may be others that were overlooked. Similarly, there may be interaction parameters that were omitted from the interaction table.

Because the approach to SOM development was conceptual, there is some chance that small numbers of object attributes and interaction parameters may have been overlooked. Another approach would have been to produce the general object model and

SOM by reviewing the Janus code. However, there are two primary benefits to using the conceptual approach. First, object model development is not restricted by the structure and implementation of the simulation code. The modeler is free to use both operational experience and knowledge of object model methodologies to produce an object-oriented representation of the simulation. This facet of the conceptual approach seemed to be particularly advantageous while working with a procedurally implemented legacy simulation such as Janus.

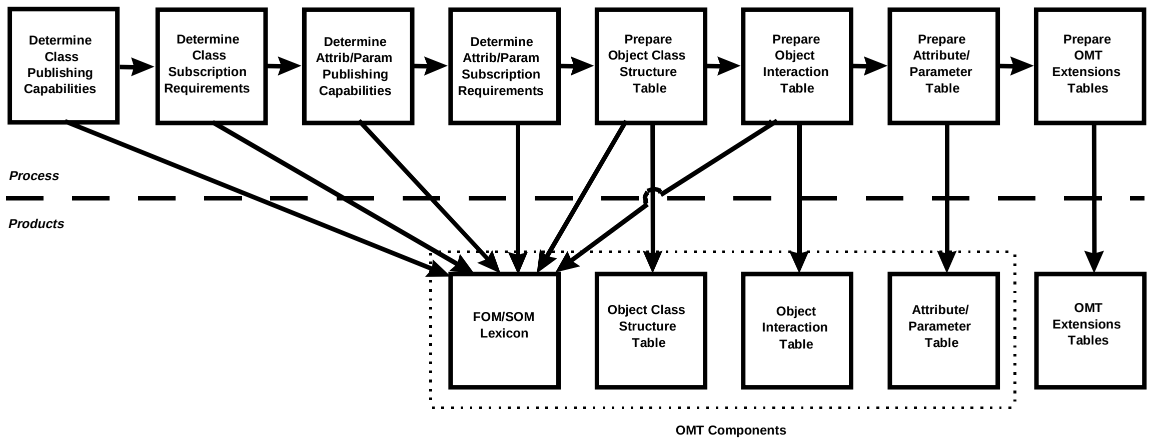
The second benefit of the conceptual approach is reduced object model development time. While Janus is well documented, the time required to gain a working knowledge of the Janus code (consisting of over two hundred thousand lines) and then to follow the code to produce an object representation would have been prohibitive.

3. Comparison with other Object Model Development Processes

Lutz [11] details a methodology to generate both SOMs and FOMs to facilitate federation interoperability. This section will compare his methodology with the General Conceptual Object Model method described in this thesis.

The eight step SOM development process proposed by Lutz is displayed in Figure 5. The following paragraphs briefly describe each step in Lutz's process.

Step 1: Determine Publishing Capabilities for Object/Interaction Classes. This step requires identification of those object and interaction classes that the model can publish. Lutz makes the point that "determination of these classes is driven by the specific intrinsic functionalities that the simulation sponsor/developer considers to be useful to and can make available to future federations."



Determine Class Publishing Capabilities
Determine Class Subscription Requirements
Determine Attrib/Param Publishing Capabilities
Determine Attrib/Param Subscription Requirements
Prepare Object Class Structure Table
Prepare Object Interaction Table
Prepare Attribute/ Parameter Table
Prepare OMT Extensions Tables
Process
Products
FOM/SOM Lexicon
Object Class Structure Table
Object Interaction Table
Attribute/ Parameter Table
OMT Extensions Tables
OMT Components

Figure 5. SOM Development Process. [11]

Step 2: Determine Subscription Requirements for Object/Interaction Classes. Lutz identifies two categories of objects/interaction classes to which the simulation may subscribe. One category contains those classes of objects and interactions that are present within the simulation and can be published as well as portrayed (reflected) in the simulation from another federate’s published data. The second category of class subscription requirements contains those classes of objects and interactions that are not internal to the simulation, but can be reflected within the simulation from data published by another federate. For example, Janus does not portray a submarine object. If another federate publishes data describing a submarine object and Janus can receive that data and portray the submarine object in the Janus scenario, then a submarine object can be added to the Janus class hierarchy.

Step 3: Determine Publishing Capabilities for Attributes/Parameters. This step identifies all attributes and parameters necessary to characterize the object and interaction classes determined in step 1. Lutz notes that “this entails specifying the complete set of public object characteristics that can be explicitly supported by the simulation, and are considered to be potentially useful in future federations" [11].

Step 4: Determine Subscription Requirements for Attributes/Parameters. This step repeats step 3 for those object and interaction classes identified in step 2.

Step 5: Prepare Object Class Structure Table. This step consists of mapping each object class identified in step 1 to an object class hierarchy in the object class structure table.

Step 6: Prepare the Object Interaction Table. The interaction class table is completed with the interaction classes identified in his steps 1 & 2 and the appropriate attributes/parameters identified in steps 3&4.

Step 7: Prepare Attribute/Parameter Table. The attribute/parameter table is completed with data identified in steps 3 & 4.

Table 4 compares the General Conceptual Object Model methodology with that presented by Lutz. One can see that the two methodologies for producing an HLA object model are similar in many respects. The primary differences are in the development of the object class structure table, the sequence in which the attribute/parameter table is completed, and in step 9. Lutz points out that "it should be noted that this suggested sequence of development activities is not the only process which can lead to efficient and robust object model construction ... many deviations from this process are possible which can lead to successful results" [11].

Step	Lutz SOM Development Process	General Conceptual Object Model Methodology
1	Determine Class Publishing Capabilities	Identify All Instantiable Objects
2	Determine Class Subscription Requirements	Not Considered in this Document*
3	Determine Attribute/Parameter Publishing Capabilities	Identify All Attributes Available to Describe Objects
4	Determine Subscription Requirements for Attributes/Parameters	Not Considered in this Document*
5	Prepare Object Class Structure Table	Build Class Hierarchy based on Common Attribute Object Groupings; Simultaneously Prepare Attribute portion of Attribute/Parameter Table
6	Prepare Object Interaction Table	Identify Interaction Parameters and Prepare Object Interaction Table; Simultaneously Prepare Parameter portion of Attribute/Parameter Table
7	Prepare Attribute/Parameter Table	Not Necessary
8	Prepare Object Model Template Extensions	Same
9	None	Reduce the General Object Model to Produce a SOM Appropriate for Federation Needs

Table 4. Comparison of Object Model Development Processes.

*The methodology in this thesis does not include consideration of either category of subscription requirements. First, the purpose of this project is to develop object models of Janus (both a general object model and a SOM) and the focus throughout has been internal to Janus**.** Second, since all publishable objects and interactions must be subscribable also, the first category of subscription requirements is already accounted for in the Janus object and interaction class structure tables. Finally, an attempt to enumerate all the possible classes of objects that may be portrayed in future federations that may be subscribable by Janus would be impractical and certainly incomplete.

B. ADDITIONAL WORK TO BE HLA COMPLIANT

Four of the five HLA rules for simulation federates rely on the SOM. SOM completion satisfies rule 6 (the first five rules apply to federations). Subsequent implementation work will satisfy the remaining four federate rules (The ten HLA rules are listed in Table 5). Distributed Janus developers at TRAC-WSMR and TRAC-MTRY must review the Janus SOM to identify potential model related implementation difficulties not anticipated by the SOM developers. This section describes the remaining work required to make Janus comply with the HLA federate rules 7 through 9; HLA federate rule 10 will not be discussed.

	Rules for federations are:
1	Federations shall have an HLA Federation Object Model (FOM), documented in accordance with the HLA Object Model Template (OMT).
2	In a federation, all representation of objects in the FOM shall be in the federates, not in the runtime infrastructure (RTI).
3	During a federation execution, all exchange of FOM data among federates shall occur via the RTI.
4	During a federation execution, federates shall interact with the runtime infrastructure (RTI) in accordance with the HLA interface specification.
5	During a federation execution, an attribute of an instance of an object shall be owned by only one federate at any given time.
	Rules for federates are:
6	Federates shall have an HLA Simulation Object Model (SOM), documented in accordance with the HLA Object Model Template (OMT).
7	Federates shall be able to update and/or reflect any attributes of objects in their SOM and send and/or receive SOM object interactions externally, as specified in their SOM.
8	Federates shall be able to transfer and/or accept ownership of attributes dynamically during a federation execution, as specified in their SOM.
9	Federates shall be able to vary the conditions (e.g., thresholds) under which they provide updates of attributes of objects, as specified in their SOM.
10	Federates shall be able to manage local time in a way which will allow them to coordinate data exchange with other members of a federation.

Table 5. HLA Rules.

1. RTI Service Interface

Janus was developed long before the concept of distributed simulation was envisioned. Therefore, there is no built-in ability to publish or receive data updates included in the Janus code. To satisfy HLA rules 7 through 9, a system must be established that provides appropriate communication between the internal Janus simulation and the HLA RTI. Currently, there are two proposed methods of establishing acceptable communications between Janus and the HLA RTI.

One proposed method is to change the Janus code to translate object updates as they occur into a format the HLA RTI understands and then output these updates to the RTI. This also requires code changes allowing Janus to receive updates from the RTI, translating them into a Janus understood format, and incorporating these updates into the Janus simulation run (See Figure 6) .

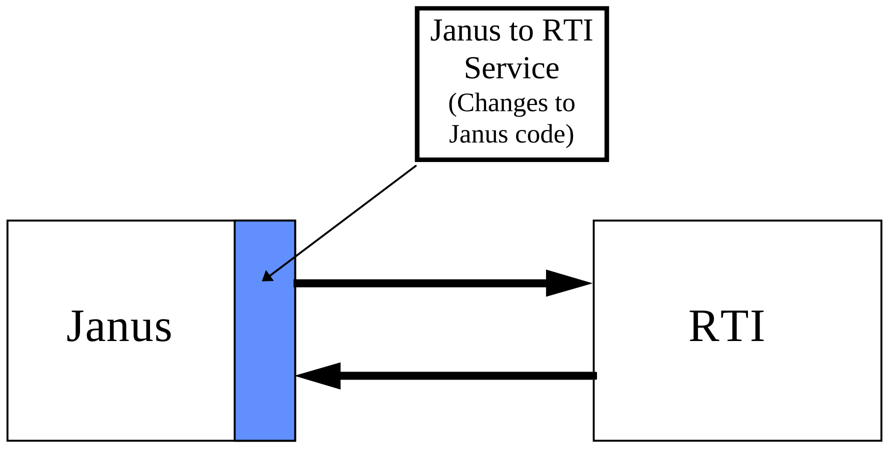


Figure 6. Janus to RTI Service Method 1.

The second proposed method of establishing communications between Janus and the RTI is to minimally change the Janus code to output object updates that occur within Janus and receive updates from the RTI that are then input to the simulation run. This method requires an additional service running in conjunction with the RTI that translates Janus updates into a format that the RTI understands and does the reverse for other federate updates coming into Janus (See Figure 7).

RTI Service Method 1 includes significant modification to the Janus code, but does not require the additional translator to run in conjunction with the RTI. RTI Service Method 2 is an extension of the J-LINK work currently underway at TRAC-MTRY using DIS architecture (J-LINK described in Chapter I). An advantage of this method is that it requires minimal revision of the internal Janus code.

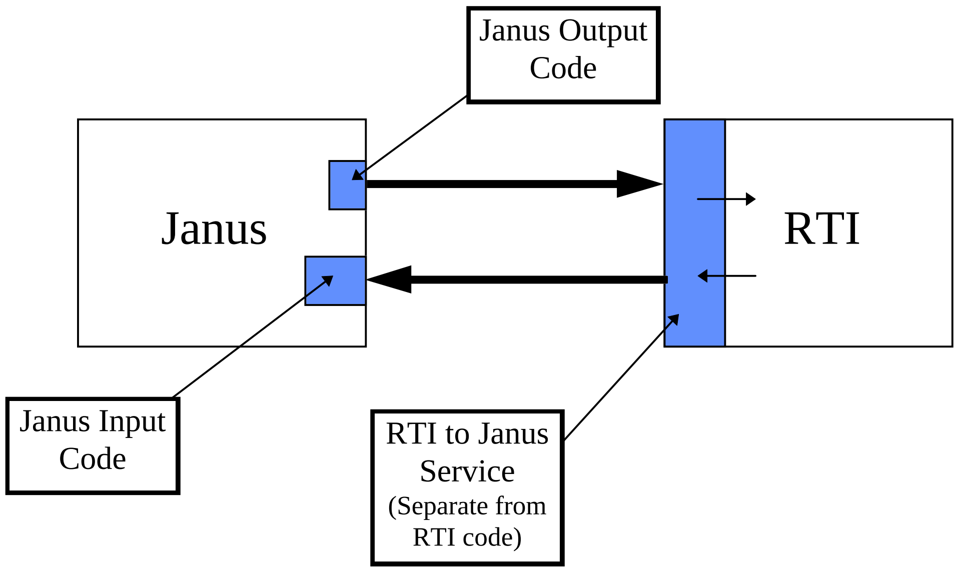


Figure 7. Janus to RTI Service Method 2.

2. Inclusion in a Federation

The final test of Janus' transition to HLA compliance will be the successful inclusion of Janus in an HLA federation. This test is likely to be six months to a year away since work has not yet begun on the Janus/RTI communication service. Current work on distributing Janus will facilitate the development of an effective communication service. TRAC-MTRY is currently considering the problem of developing an RTI Service for Janus based on the SOM developed through this research.

C. CHAPTER SUMMARY

Chapter II provided the reader with a detailed view of the General Conceptual Model methodology. The synopsis in this chapter was intended to provide the reader with a concisely packaged view of the process. It also serves as the starting point for a look at the results of using this approach to create a HLA SOM.

There is a potential danger incurred by using this methodology. Since the simulation code is not used to derive the model, there may be errors of omission.

However, this danger must be weighed against the benefit obtained by the conceptual approach. To eliminate the danger of omitting critical aspects of the simulation, the general object model will be reviewed by Janus code experts to verify that the model accurately represents the Janus simulation.

The Lutz [11] object model development process provides an opportunity to compare the General Conceptual Object Model methodology with another proposed procedure to create a HLA SOM. When comparing the two methods, it is important to remember that the Lutz process is intended to work with any simulation model. Conversely, the General Conceptual Model methodology specifically addresses procedurally coded legacy simulations developed prior to the advent of distributed technology.

The work necessary to deliver a fully HLA capable Janus is just beginning. In addition to the SOM, it will be necessary to build a RTI service that will allow Janus to communicate through the HLA RTI with other federates. Current work within the TRAC organization to develop a DIS capable Janus could facilitate the production of a RTI service.

IV. CONCLUSIONS AND RECOMMENDATIONS

A. SUMMARY

This section provides conclusions and recommendations for model proponents who desire to bring their legacy simulations into compliance with the HLA requirements. The research and experience resulting from this thesis clearly indicates that it is possible for procedural legacy simulations, and Janus in particular, to comply with the HLA rule requiring a SOM. In order to achieve compliance with this rule, it is advantageous to first develop a complete conceptual model of the legacy simulation. Finally, if the model will be used in a federation requiring analysis of model outputs, an analyst should be included during SOM development.

B. JANUS COMPLIANCE WITH THE SOM REQUIREMENT

The research conducted to produce this document indicates that Janus can certainly meet the HLA SOM requirement. The SOM is a significant step toward transitioning Janus into HLA compliance. Additional steps for Janus to conform to the remaining HLA requirements are likely to be more costly. SOM development for Janus provided significant insight into the size of these costs by providing a means to identify the additional steps necessary to make Janus HLA compliant. A decision to proceed with additional work to produce an HLA compliant Janus model must consider these costs. In this case, the HLA SOM provides a feasibility gauge that can be used to measure the cost and complexity of further HLA investment in Janus. Legacy model proponent organizations may benefit from detailed review of their simulations for possible compliance with HLA. Beginning this review by producing a fully developed SOM at relatively low cost will aid the proponent in determining the cost effectiveness of further investment in HLA compliance.

C. ADVANTAGES OF THE GENERAL OBJECT MODEL

In many cases it will be appropriate to produce a general conceptual model of a legacy simulation prior to producing an HLA SOM. Building the general conceptual object model of Janus prior to producing the SOM required more time and detail than simply building the SOM. This section explains the reasons why it was appropriate to spend the additional resources to produce the general model.

1. Facilitate Model/Federate use in Analysis

As described in chapter I, one of Janus' great strengths is the flexibility to model many different platforms. The Janus database provides hundreds of attributes to specify the functionality and characteristics of many types of militarily important platforms. A primary benefit of the general object model is that it provides the military analyst with a full representation of Janus' ability to model objects. Conversely, the HLA SOM is a subset of the general object model that may include only small portions of the general model. An analyst attempting to identify an appropriate model to include in a federation needs to find a simulation that will allow him to quantify some specific MOE. The analyst may overlook a potential federate as a candidate if using only the SOM to identify the model's capabilities, but may identify an appropriate simulation through its general object model.

2. Avoid Omitting Valuable/Critical Aspects of the Model

Since Janus has no inherent object representation and it was not practical to produce a SOM by reviewing the Janus code, the general conceptual model provided a means to minimize the possibility that an important piece of Janus was not overlooked. As pointed out earlier, the methodology chosen does not guarantee that the general object model will perfectly describe Janus prior to the review by TRAC-WSMR. However, producing a thoroughly researched general object model increases the chances that all relevant information is included in the SOM prior to any external review.

3. Capture Janus Conceptually for Development of Future Models

While Janus is still widely used both as an analysis tool and for training, even avid Janus supporters have come to realize that Janus is likely near the end of its service life. The U.S. Army is actively seeking a replacement model for Janus. In this light, the general object model of Janus provides an object-oriented look at the minimum standard for a Janus replacement. Additionally, if deemed appropriate, the general object model of Janus could be used as a starting point for the development of an entirely new model.

D. INCLUDE AN ANALYST IN SOM DEVELOPMENT

It is advantageous to include an analyst during SOM development, rather than only a programmer or implementer. As a minimum, the analyst should be an active member of any SOM development team. This is more important if the simulation model is to be used as an analysis tool.

The programmer or implementer can certainly produce an object model of a given model quickly and efficiently. However, it is the analyst who must use the model to quantify measures of effectiveness. As pointed out by Lutz, the model proponent has significant latitude in what is included in the SOM based the projected use of the model [11]. The analyst brings to the SOM development process an understanding of the kind of studies in which the model may be included, what measures of effectiveness the model may be expected to quantify, and therefore what is important to include in the SOM. Chapter III, paragraph E provides some understanding of the types of issues an analyst must consider as he determines what should be included in a SOM.

E. DISCUSSION

The software tool used to document the Janus general object model and SOM for this research was the AEgis Research Corporation Object Model Development Tool (OMDT) [16]. The OMDT was developed by AEgis Research Corporation at the request of the DMSO to streamline the process of documenting HLA object models. While the OMDT provides a convenient means of capturing the Janus object models in the DMSO specified format, it does not allow manipulation of the tables for output. Consequently, the printed output from the OMDT cannot be scaled to fit presentably in hard copy. A Pascal program provided the means to overcome this deficiency and provide examples of the general object model tables in this document. The Pascal code for this program is included at Appendix B.

F. CONCLUSIONS AND FUTURE WORK

The significant result of this research is the methodology to produce a HLA Simulation Object Model for procedural legacy simulations. Currently, two methodologies have been presented for this purpose: that proposed by Lutz as a generic approach applicable to all models [11]; and the General Conceptual Object Model method for procedurally coded models presented in this thesis. These two methodologies were compared in Chapter III. At some point, others will develop methodologies appropriate for specific model types or perhaps, like Lutz, appropriate for a wide range of models including those coded using either procedural or object-oriented languages. As work in the SOM development field continues, periodic reviews of all proposed processes will produce refinements in the General Conceptual Object Model methodology.

The effort to bring Janus into compliance with the HLA standards will continue at TRAC-MTRY. After the object models are reviewed by TRAC-WSMR and a potential RTI service implementer, TRAC-MTRY has tentative plans to begin work on the RTI service. This project has generated significant interest within the TRAC organization as a viable means to extend the use of Janus into the HLA environment. Other parallel work includes a proposal to develop a proto-type analysis federate at TRAC-MTRY and use the HLA capable Janus to validate the analysis federate concept.

The DoD has invested many resources toward the development and implementation of the HLA. The HLA concept was validated in the summer of 1996 using proto-federations produced for that purpose. While the concept proved viable, there is much work remaining before HLA can be considered matured. This thesis has identified some difficulties with modifying a legacy simulation so that it can operate using the HLA. Some larger difficulties are yet to be solved.

One such problem is the federate object representation of platforms. The object representation of entities, elements of nature, and man-made objects which is fundamental to the HLA could result in difficulties in federation development. If similar platforms from two different federates have significantly different object representations, it may not be worth the trouble to modify one or both potential federates to achieve an acceptable level of interoperability. As an example, suppose one potential federate portrays a smoke cloud as a series of geometric shapes (trapezoids) with 16 attributes. The other potential federate portrays a smoke cloud as circle with some radius and height (two attributes). It will require a detailed subroutine in one of the federates to translate between the two different object representations of the smoke cloud so that it is depicted accurately in both federates. In this case, even if the subroutine is added so that both federates can portray the smoke cloud, it is obvious that they are both not really “seeing” the same smoke cloud (one is a circle or cylinder while the other is a series of trapezoids). The Army is addressing this problem with a project called the Standard Army Model and Simulation Objects Study. The project’s goal is to identify standard object representations of those objects most likely to be used in future Army simulations thereby avoiding conflicts in object representation in future simulation models.

While the HLA promises to allow significantly broader interoperability between simulation models, the future will reveal the true benefit of the HLA. The shortcomings in DIS that lead to the HLA were not apparent during the early years of DIS. In the same manner significant flaws in the HLA concept may not become visible for several years.

APPENDIX A. Examples of Completed Object Model Tables

Object Class Structure Table

Object Interaction Table

Attribute/Parameter Table

Attribute/Parameter Table

Attribute/Parameter Table

Attribute/Parameter Table

Attribute/Parameter Table

Attribute/Parameter Table

Attribute/Parameter Table

Attribute/Parameter Table

Attribute/Parameter Table

APPENDIX B. PASCAL CODE FOR ATTRIBUTE/PARAMETER TABLE

The AEgis Research Object Model Development Tool (OMDT) facilitates production of object model tables in the appropriate HLA OMT format. However, the OMDT does not allow manipulation of the table for output. Consequently, the printed output from the OMDT cannot be scaled to fit presentably in hardcopy.

The OMDT file can be opened as a text file. A typical section of the OMDT file is shown at Listing 1 in the text file format.

```
(Class (Name "Base_Platform")
        (PSCapabilities S)
        (Description "Superclass of all objects that will comprise the military forces participating in a planned
simulation")

    (Attribute (Name "MovementDirection")
              (DataType "any")
              (Cardinality "1")
              (Accuracy "perfect")
              (AccuracyCondition "always")
              (UpdateType Conditional)
              (TransferAccept N)
              (UpdateReflect UR)
              (DeliveryCategory RELIABLE)
              (MessageOrdering TIMESTAMP)
    )
    (Attribute (Name "Location")
              (DataType "any")
              (Cardinality "1")
              (Accuracy "perfect")
              (AccuracyCondition "always")
              (UpdateType Conditional)
              (TransferAccept N)
              (UpdateReflect UR)
              (DeliveryCategory RELIABLE)
              (MessageOrdering TIMESTAMP)
    )
```

Listing 1. Example of OMDT file in Text Format.

The following Pascal program was used to produce a text file from the OMDT file that could be opened in Microsoft Excel and imported to Microsoft Word. The hard copy output is seen at APPENDIX A:

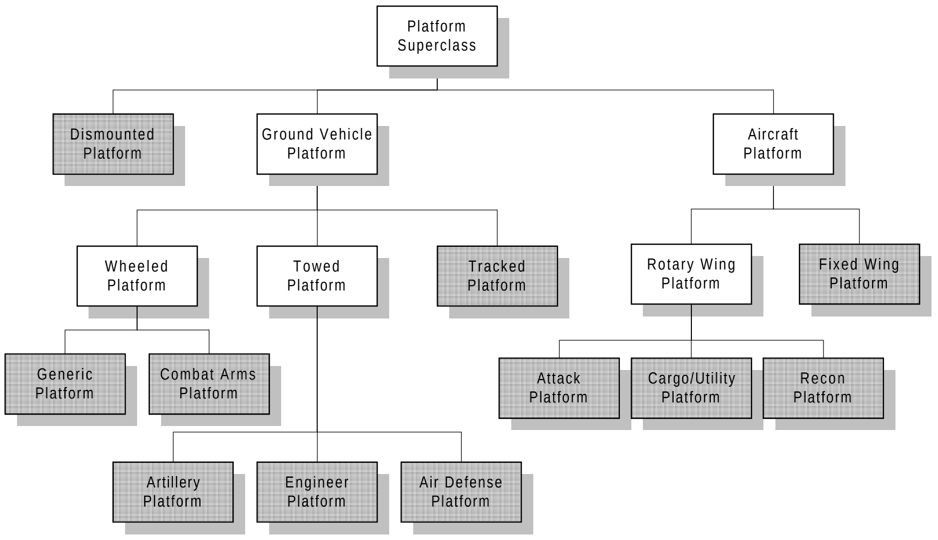
```
{*************************************************}
{                                                 }
{              Larimer, Larry R.                  }
{              Thesis                             }
{              OMDT Output Tool                   }
{                                                 }
{  This program reads an AEgis Research OMDT file }
{  (*.omd) and creates a text file in HLA         }
{  Attribute/Parameter table format.              }
{                                                 }
{*************************************************}

Program OMDTOutput;

Uses Wincrt;

Type LineType = String[255];

Var Line,
    Temp,
    Whole:        LineType;
    Inputfile,
             Outputfile:  String[40];

    Infile,
    Outfile:     Text;
    Stop:        Integer;

Procedure Attribute(Var Input,Output:Text;Var WholeP: LineType);

Var Tab:  String[1];
    Count,
    StopP:Integer;
    TempP,
    LineP,
    Attrib: LineType;

Begin
    Attrib:='';
    Tab:=#9;
    Attrib:=Attrib+WholeP;
    Readln(Input,LineP); {DataType}
    LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    StopP:=Pos(')',LineP);
    TempP:=Copy(LineP,12,StopP-12-1);
    Attrib:=Attrib+Tab+TempP;
    Readln(Input,LineP); {Cardinality}
    LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    StopP:=Pos(')',LineP);
    TempP:=Copy(LineP,15,StopP-15-1);
    Attrib:=Attrib+Tab+TempP;
    Readln(Input,LineP);
    LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    If(LineP[2]='U') and (LineP[3]='n') then begin  {Units}
      StopP:=Pos(')',LineP);
      TempP:=Copy(LineP,9,StopP-9-1);
      Attrib:=Attrib+Tab+TempP;
      Readln(Input,LineP);
      LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    End
    Else begin
    Attrib:=Attrib+Tab;
    End;
    If (LineP[6]='l') and (LineP[10]='o') then begin {Resolution}
      StopP:=Pos(')',LineP);
      TempP:=Copy(LineP,14,StopP-14-1);
      Attrib:=Attrib+Tab+TempP;
      Readln(Input,LineP);
      LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    End
    Else begin
      Attrib:=Attrib+Tab;
    End;
    If (LineP[6]='r') and (LineP[10]=' ') then begin {Accuracy}
      StopP:=Pos(')',LineP);
      TempP:=Copy(LineP,12,StopP-12-1);
      Attrib:=Attrib+Tab+TempP;
      Readln(Input,LineP);
      LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    End
    Else begin
      Attrib:=Attrib+Tab;
    End;
    If (LineP[6]='r') and (LineP[10]='C') then begin {Accuracy Condition}
      StopP:=Pos(')',LineP);
      TempP:=Copy(LineP,21,StopP-21-1);
      Attrib:=Attrib+Tab+TempP;
      Readln(Input,LineP);
      LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    End
    Else begin
      Attrib:=Attrib+Tab;
    End;
    If (LineP[6]='t') and (LineP[10]='p') then begin {UpdateType}
      StopP:=Pos(')',LineP);
      TempP:=Copy(LineP,13,StopP-13);
      Attrib:=Attrib+Tab+TempP;
      Readln(Input,LineP);
      LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    End
    Else begin
      Attrib:=Attrib+Tab;
    End;
    If (LineP[6]='t') and (LineP[10]='n') then begin {UpdateCondition}
      StopP:=Pos(')',LineP);
      TempP:=Copy(LineP,19,StopP-19-1);
      Attrib:=Attrib+Tab+TempP;
      Readln(Input,LineP);
      LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    End
    Else begin
      Attrib:=Attrib+Tab;
    End;
    If (LineP[6]='s') and (LineP[10]='A') then begin {TransferAccept}
      StopP:=Pos(')',LineP);
      TempP:=Copy(LineP,17,StopP-17);
      Attrib:=Attrib+Tab+TempP;
      Readln(Input,LineP);
      LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    End
    Else begin
      Attrib:=Attrib+Tab;
    End;
    If (LineP[6]='t') and (LineP[10]='f') then begin {UpdateReflect}
      StopP:=Pos(')',LineP);
      TempP:=Copy(LineP,16,StopP-16);
      Attrib:=Attrib+Tab+TempP;
      Readln(Input,LineP);
      LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    End
    Else begin
      Attrib:=Attrib+Tab;
    End;
    If (LineP[6]='r') and (LineP[10]='i') then begin {Description}
```

```
     Readln(Input);
    End;
    If (LineP[6]='v') and (LineP[10]='C') then begin {DeliveryCategory}
      StopP:=Pos(')',LineP);
      TempP:=Copy(LineP,19,StopP-19);
      Attrib:=Attrib+Tab+TempP;
      Readln(Input,LineP);
      LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    End
    Else begin
      Attrib:=Attrib+Tab;
    End;
    If (LineP[6]='a') and (LineP[10]='r') then begin {MessageOrdering}
      StopP:=Pos(')',LineP);
      TempP:=Copy(LineP,18,StopP-18);
      Attrib:=Attrib+Tab+TempP;
      Readln(Input);
    End;
    Writeln(Output,Attrib);
End;

Procedure Parameter(Var Input,Output:Text;Var WholeP: LineType);

Var Tab:  String[1];
    Count,
    StopP:Integer;
    TempP,
    LineP,
    Parameter: LineType;

Begin
    Parameter:='';
    Tab:=#9;
    Parameter:=Parameter+WholeP;
    Readln(Input,LineP);
    LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    If(LineP[2]='O') and (LineP[3]='r') then begin {Order}
      Readln(Input,LineP);
      LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    End;
    If (LineP[6]='T') and (LineP[10]=' ') then begin {DataType}
      StopP:=Pos(')',LineP);
      TempP:=Copy(LineP,12,StopP-12-1);
      Parameter:=Parameter+Tab+TempP;
      Readln(Input,LineP);
      LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    End
    Else begin
      Parameter:=Parameter+Tab;
    End;
    If (LineP[6]='i') and (LineP[10]='i') then begin {Cardinality}
      StopP:=Pos(')',LineP);
      TempP:=Copy(LineP,15,StopP-15-1);
      Parameter:=Parameter+Tab+TempP;
      Readln(Input,LineP);
      LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    End
    Else begin
      Parameter:=Parameter+Tab;
    End;
      If(LineP[2]='U') and (LineP[3]='n') then begin {Units}
      StopP:=Pos(')',LineP);
      TempP:=Copy(LineP,9,StopP-9-1);
      Parameter:=Parameter+Tab+TempP;
      Readln(Input,LineP);
      LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    End
    Else begin
    Parameter:=Parameter+Tab;
    End;
    If (LineP[6]='l') and (LineP[10]='o') then begin {Resolution}
      StopP:=Pos(')',LineP);
      TempP:=Copy(LineP,14,StopP-14-1);
      Parameter:=Parameter+Tab+TempP;
      Readln(Input,LineP);
      LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    End
    Else begin
      Parameter:=Parameter+Tab;
    End;
    If (LineP[6]='r') and (LineP[10]=' ') then begin {Accuracy}
      StopP:=Pos(')',LineP);
      TempP:=Copy(LineP,12,StopP-12-1);
      Parameter:=Parameter+Tab+TempP;
      Readln(Input,LineP);
      LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    End
    Else begin
      Parameter:=Parameter+Tab;
    End;
    If (LineP[6]='r') and (LineP[10]='C') then begin {AccuracyCondition}
      StopP:=Pos(')',LineP);
      TempP:=Copy(LineP,21,StopP-21-1);
      Parameter:=Parameter+Tab+TempP;
      Readln(Input,LineP);
      LineP:=Copy(LineP,Pos('(',LineP),Length(LineP)-Pos('(',LineP)+1);
    End
    Else begin
      Parameter:=Parameter+Tab;
    End;
    Writeln(Output,Parameter);
End;


{*********************Driver***************************}

Begin
  Writeln('What is the input file name?:');
  Readln (Inputfile);
  Assign(Infile,Inputfile);
  Reset(Infile);
  Writeln('What is the output file name?:');
  Readln(Outputfile);
  Assign(Outfile,Outputfile);
  Rewrite(Outfile);
  While not SEEKEOF(Infile) Do Begin
    Line:='';
    Readln(Infile,Line);
    Line:=Copy(Line,Pos('(',Line),Length(Line)-Pos('(',Line)+1);
    If (Line[6]='s') and (Line[10]='a') and (Line[2]='C') then begin{Class Name}
      Stop:=Pos(')',Line);
      Temp:=Copy(Line,15,Stop-15-1);
      Whole:=Temp;
    End;
    If (Line[6]='i') and (Line[10]='e') then begin {Attribute}
      Stop:=Pos(')',Line);
      Temp:=Copy(Line,19,Stop-19-1);
      Whole:=Whole+#9+Temp;
      Attribute(Infile,Outfile,Whole);
      Whole:='';
    End;
    If (Line[6]='r') and (Line[10]='i') and (Line[2]='I') then begin
{Interaction Name}
      Stop:=Pos(')',Line);
      Temp:=Copy(Line,21,Stop-21-1);
      Whole:=Temp;
    End;
    If (Line[6]='m') and (Line[10]='r') then begin {Parameter}
      Stop:=Pos(')',Line);
      Temp:=Copy(Line,19,Stop-19-1);
      Whole:=Whole+#9+Temp;
      Parameter(Infile,Outfile,Whole);
      Whole:='';
    End;
  End;
  Close(Infile);
  Close(Outfile);
End.
```

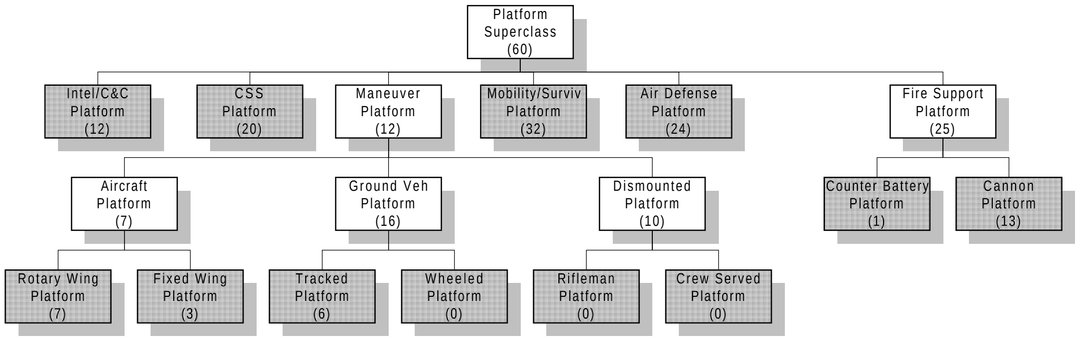
APPENDIX C. JANUS OBJECT CLASS HIERARCHY

The following diagrams outline the superclasses and subordinate class hierarchies identified in Janus:

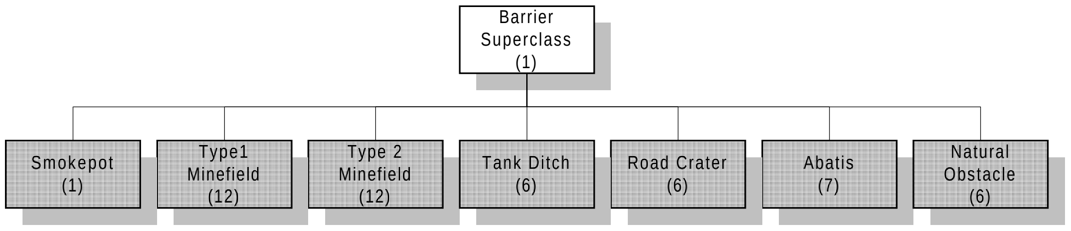
NOTE: Shaded leaves indicate instantiable objects. Numbers in parenthesis indicate attributes.

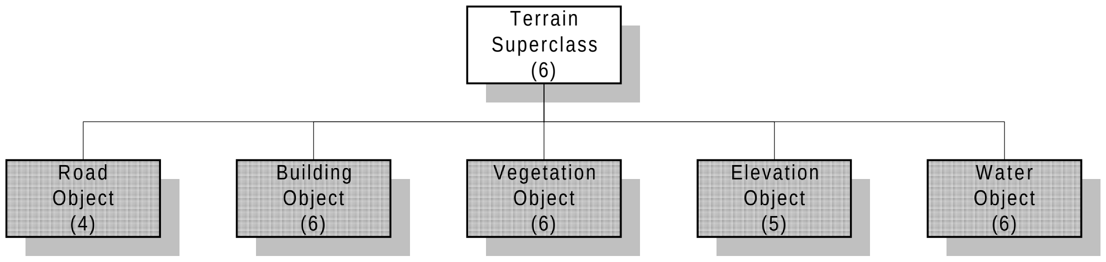


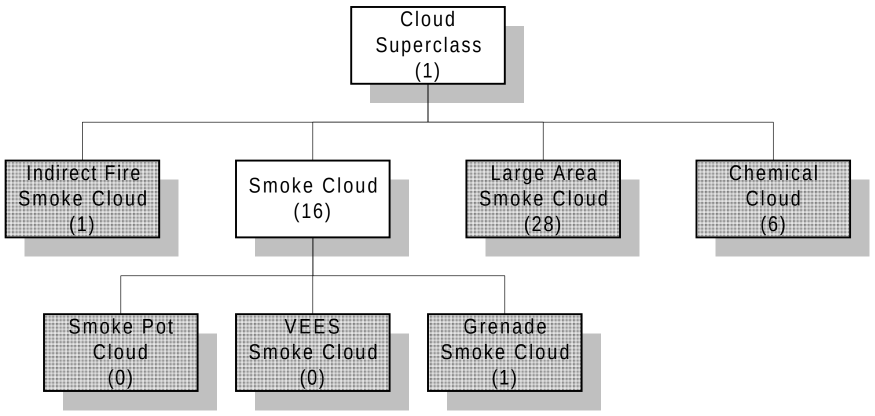
Original Platform Class Hierarchy

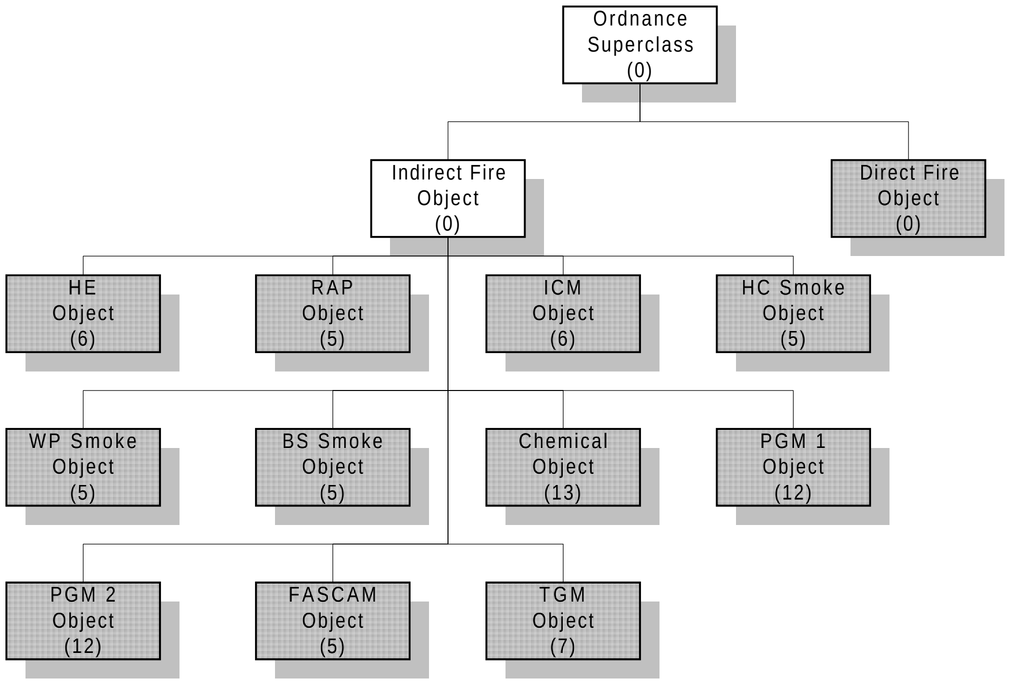


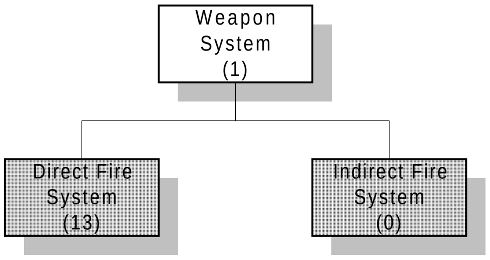
Modified Platform Class Hierarchy

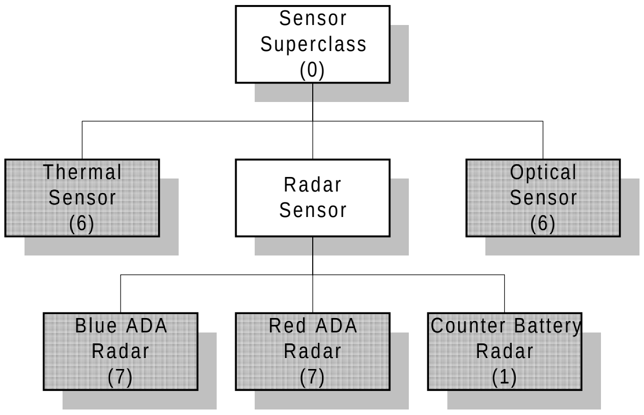



Terrain Superclass (6)
Road Object (4)
Building Object (6)
Vegetation Object (6)
Elevation Object (5)
Water Object (6)

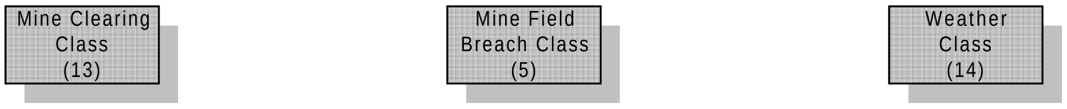

Cloud Superclass (1)
Indirect Fire Smoke Cloud (1)
Smoke Cloud (16)
Large Area Smoke Cloud (28)
Chemical Cloud (6)
Smoke Pot Cloud (0)
VEES Smoke Cloud (0)
Grenade Smoke Cloud (1)


Ordnance Superclass (0)
Indirect Fire Object (0)
Direct Fire Object (0)
HE Object (6)
RAP Object (5)
ICM Object (6)
HC Smoke Object (5)
WP Smoke Object (5)
BS Smoke Object (5)
Chemical Object (13)
PGM 1 Object (12)
PGM 2 Object (12)
FASCAM Object (5)
TGM Object (7)





The following three superclass objects have no subordinate hierarchy:



APPENDIX D. JANUS TARGET ACQUISITION ALGORITHM

This appendix explains the Janus target acquisition algorithm to give the reader a better understanding of the process required to identify dynamic attributes and parameters.

A. GENERAL

Janus uses a common target acquisition algorithm often referred to as the Night Vision Electro-Optical Laboratory (NVEOL) Model. The model requires inputs from the prospective target platform, atmospheric objects, and the sensor object conducting the search. [13]

The NVEOL target acquisition model is probabilistic. The probability of target detection is based on the number of resolvable cycles a sensor can distinguish across the minimum dimension of the target and the amount of time the target is in the sensor field of view.

B. EXPLANATION OF RESOLVABLE CYCLES

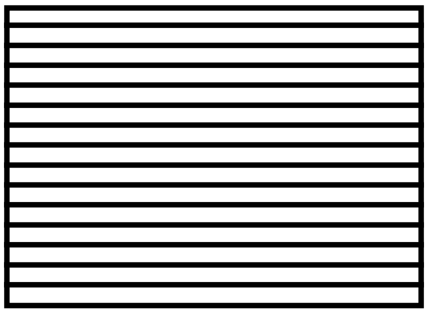


Figure D-1.

When looking at the box with horizontal lines in Figure D-1 one can easily distinguish the number of white and black lines. The distance between black lines is analogous to the wavelength of one “cycle” in the NVEOL model. As the lines in the figure are brought closer together, it becomes increasingly difficult to distinguish between them. This is illustrated by comparing Figure D-1 to Figure D-2. At some point, as the lines are further compressed, the sensor is no longer capable of distinguishing between the lines. The figure becomes a blur of washed out lines.

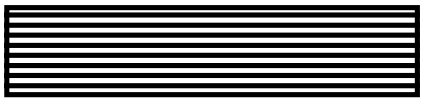


Figure D-2.

The minimum cycle length that the sensor can distinguish defines the sensitivity or capability of the sensor [15].

C. VARIED CONTRAST

This minimum cycle length varies with the contrast between the target and the background.   A sensor that can distinguish a cycle length of 10 centimeters when the lines are black and white may require 20 centimeters when the lines are different shades of gray as in Figure D-3.

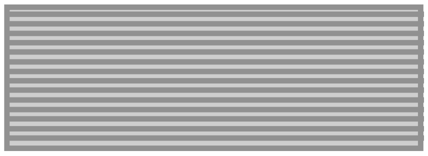


Figure D-3.

D. MEASURE OF SENSOR PERFORMANCE

The model accounts for the size and range of the target by specifying the number of cycles that are produced across the targets minimum dimension. Because the number of cycles will decrease as range increases, the number of cycles measured across the target is defined by the angle θ shown in Figure D-4. Note that for any given angle θ

there is a fixed number of cycles that will be viewed by the sensor regardless of the range to the target.

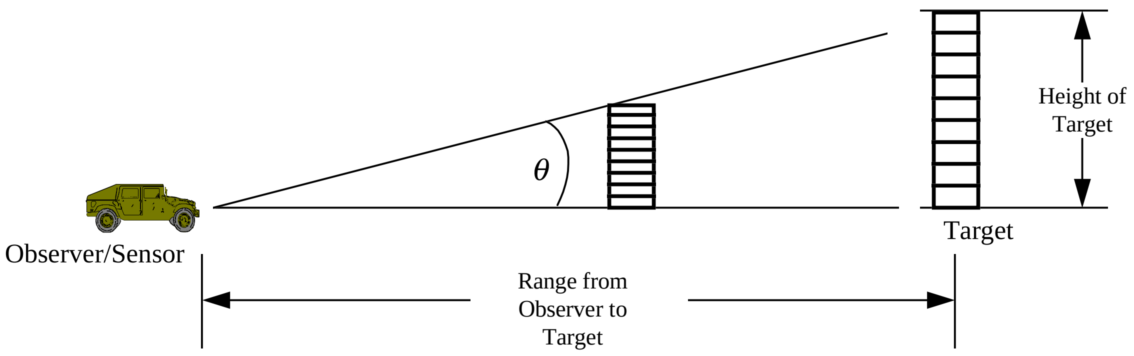


Figure D-4.

The sensitivity of the sensor is defined for use in the model as the number of cycles per milliradian that the sensor can distinguish at a given contrast level.

$$Spacial\ frequency = \frac{Cycles}{miliradians}$$

Where spatial frequency is cycles per milliradian, and Cycles is the cycle length in meters. For example, if the cycles per milliradian is 1, and the angle from the sensor across the target is 9 milliradians, the sensor will distinguish 9 cycles across the target. See angle θ_1 at Figure D-5. If the target is further from the sensor, the angle θ will be smaller, and a fewer number of cycles will cross the target. See angle θ_2 at Figure D-5.

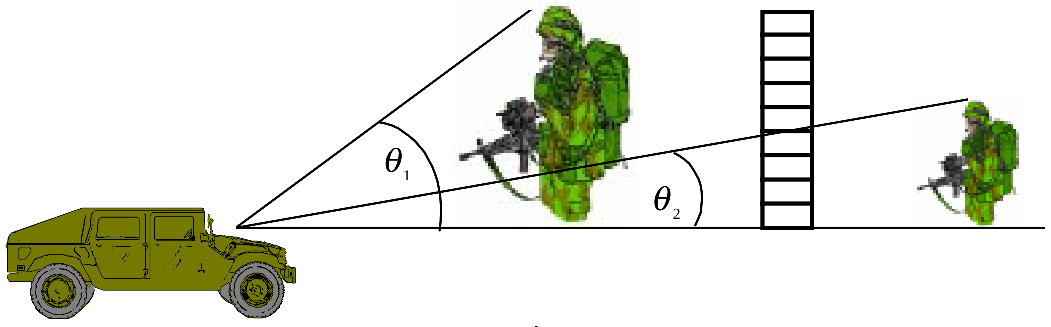


Figure D-5.

Since the angle θ is 1 milliradian when the target dimension is 1 meter and range is 1000 meters, θ is easily approximated with the following formula:

$$\theta = \frac{target\,dimension}{target\,range}$$

Where target dimension is in meters and target range is in kilometers. Note that this is specific to Janus, some models measure target range in meters. The units used for range are dependent upon how the model is parameterized.

Given the angle θ and the number of cycles per milliradian that the sensor can distinguish, the next step is to determine the number of resolvable cycles across the target. This is calculated as follows:

$$\#cycles = \theta\left(\frac{cycles}{miliradian}\right)$$

This result (#cycles) is how many resolvable cycles the sensor will distinguish across the target. The more resolvable cycles, the greater the chance of detection. See Figure D-6 below. [14]

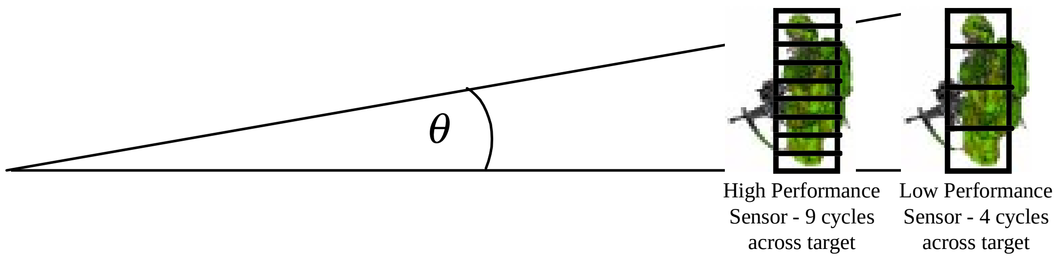

θ
High Performance Sensor - 9 cycles across target
Low Performance Sensor - 4 cycles across target

Figure D-6.

E. SIGNATURE AT THE TARGET

The contrast between the target and its background form a ratio which is called the target signature. For optical sensors, this ratio is as follows:

$$target\,signature = \frac{|target\,brighness - background\,brightness|}{background\,brightness}$$

Target brightness and background brightness are in lumens (a measure of the light energy reflected off a surface). This ratio defines the signature of the target at the target location and obviously takes on values greater than 0. A very large target signature value indicates distinct contrast between target and background (essentially black and white). As the target signature value approaches 0, the target and background blend together. When the target signature equals 0, the target is indistinguishable from the background. [15]

F. ATTENUATION IN THE ATMOSPHERE

The target signature must travel through the atmosphere to reach a searching sensor. The target signature is affected by normal atmospheric conditions as it moves along the line of sight between the target and the sensor. Additionally, the target signature may be affected by one or more cloud objects. See Figure D-7.

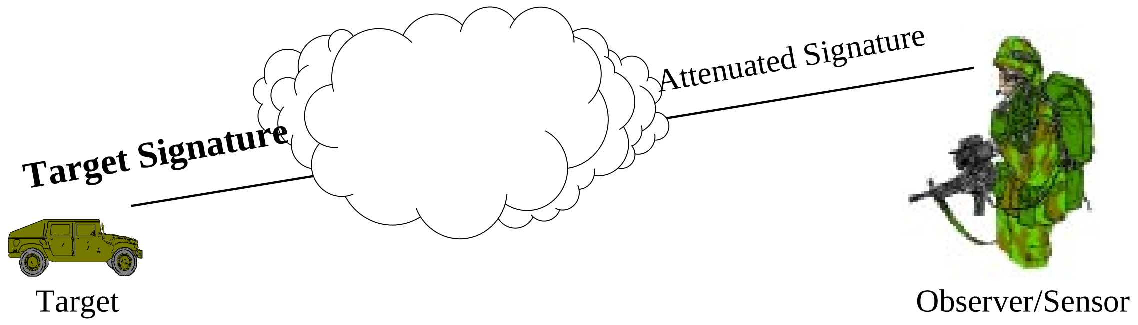


Figure D-7.

The attenuated signature that arrives at the target is characterized by the following equation:

$$a_S = t_S \times a_A \times a_C$$

a_S - is the attenuated signature that arrives at the sensor.

t_S - is the target signature (contrast).

a_A - is the attenuation due to normal atmospheric and solar conditions.

a_C - is the attenuation caused by any smoke cloud that may be between the sensor and the target.

There are two attenuation effects for normal weather considered in the Janus target acquisition algorithm. These are attenuation caused by the air as the image is transmitted to the sensor and the solar effects. Solar effect is the degrading of the target signature due to the location and angle of the sun. These effects are incorporated into one formula resulting in the "transmission" or attenuation due to normal atmosphere as follows [14]:

$$a_A = \frac{1}{1 + SGR * \left(e^{(\alpha(R))} - 1\right)} \qquad (1)$$

Where

a_A- is the attenuation due to normal atmospheric conditions.

SGR- is the sky-to-ground-ratio - a measure of the target image dispersion due to the angle of the sun and interaction between the sun and any cloud cover.

α - is the attenuation caused by the atmosphere (air density, air motion, etc.).

R - is the range from the target to the sensor.

The atmospheric attenuation always takes on values between 0 and 1. Note that as *R, SGR,* or α increase while holding the other two variables constant, the value of atmospheric attenuation decreases. [14]

Attenuation due to one or more smoke clouds between the sensor and target is modeled by the a_C component of the attenuated signature equation. When Janus checks for line of sight between the sensor and target any smoke clouds between the two are included in the a_C component. a_C, like the a_A takes on values between 0 and 1. Dense or thick smoke produces a low value for a_C, while thin smoke produces values of a_C close to 1.

One can see from the attenuated signature equation (equation 1) that if the value of both a_C and a_A are 1 then the attenuated target signature = the target signature. Conversely, if the values of a_A and a_C are both small, a significantly reduced signature will arrive at the sensor.

As described previously, the NVEOL model measure of sensor capability is the minimum resolvable cycle that the sensor can distinguish. The minimum resolvable cycles is quantified using the cycles per milliradian or spatial frequency. Also discussed previously, the spatial frequency for a given sensor varies with the contrast between the target and the background. Sensor performance data is stored in the Janus database as minimum resolvable contrast/spatial frequency pairs. Twenty such data pairs, with straight line interpolation between pairs, form a curve called the Minimum Resolvable Contrast (MRC) curve (Figure D-8).

The attenuated target signature is entered in the MRC curve and the corresponding spatial frequency (cycles/mr) is calculated. As illustrated in Figure D-8, an attenuated signature of 0.3 results in a spatial frequency of approximately 1.657.

Recall from paragraph D that the number of cycles the sensor can distinguish across the target is calculated first by determining the angle θ from the target minimum dimension and range, then by taking the product of θ and the spatial frequency. (Hoock)

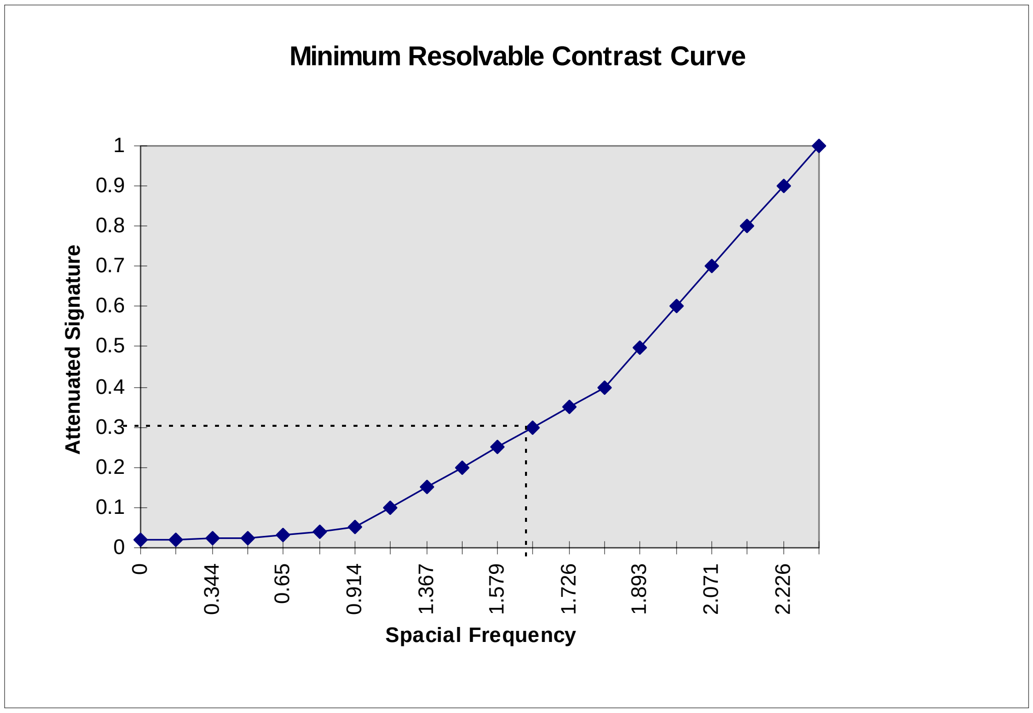


Figure D-8.

Following the example from the previous paragraph, assume a sensor has received an attenuated contrast of .3 and the associated spatial frequency is 1.657 cycles per milliradian. Given that the angle θ is 2 milliradians as in Figure D-9, then the number of cycles the sensor can distinguish across the target is 3.314.

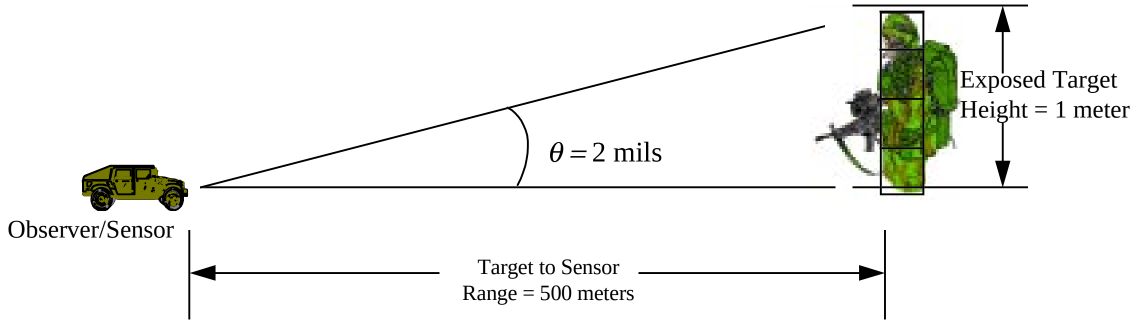


Figure D-9.

G. DETERMINING PROBABILITY OF TARGET DETECTION

The probability that the target will be detected is a function of the number of cycles across the target. Experimentation has shown that the fraction of observers who will eventually detect a specific target given an infinite amount of search time can be modeled as follows:

$$P_\infty = R^{\left(\frac{E}{\left(1+R^E\right)}\right)} \tag{2}$$

Where R is the ratio $\frac{N}{N50}$, and $E = 2.7 + 0.7\left(\frac{N}{N50}\right)$

N = the number of cycles across the target (3.314 in the previous example).

N50 is the number of cycles across the target corresponding to a .5 probability of target detection given an infinite amount of search time. Janus uses 1.0 for the N50 value. [13]

Once the value for P_∞ is determined, Janus generates a uniform random number on the interval 0 to 1 and compares the random number to P_∞. If the random number is less than P_∞, then the target is further considered for detection. If it is greater than P_∞ then the target is no longer considered for detection.

After the target passes the P_∞ test for detection, Janus uses the exponential distribution to model the probability of target detection as a function of time. The P_∞ value is used to estimate the rate of detection for this exponential distribution in the following way:

$$\lambda = \begin{cases} \frac{P_\infty}{3.4}, & for\,\frac{N}{N50} \le 2 \\ \frac{N}{\left(6.8\left(N50\right)\right)}, & for\,\frac{N}{N50} > 2 \end{cases} \tag{3}$$

This value of λ, entered in the standard exponential cumulative distribution function results in the following probability of detection as a function of time:

$$P_{Detect}(t) = 1 - e^{-\lambda t} \tag{4}$$

where t is the amount of time the target has spent in the sensor’s field of view.

Each target detection cycle Janus evaluates the probability of detection against new uniform [0,1] random number until the target is either detected or is no longer available for detection (target dies, mounts another platform, moves out of detection range, etc.). [14]

APPENDIX E. JANUS TUTORIAL

JANUS TUTORIAL 1

1. INTRODUCTION

The amphibious task force commander clears his throat and everyone comes to order. The commander begins, "You've all been briefed on the situation. Make sure you understand my intent. I want a deep helicopter strike to the south followed by an amphibious landing in the center supported by naval gunfire. In the north, rotary and fixed wing air assets will conduct a supporting attack.,…" Commanders and staffs can train for such a scenario using Janus, a legacy simulation which possesses many proven features.

2. JANUS OVERVIEW

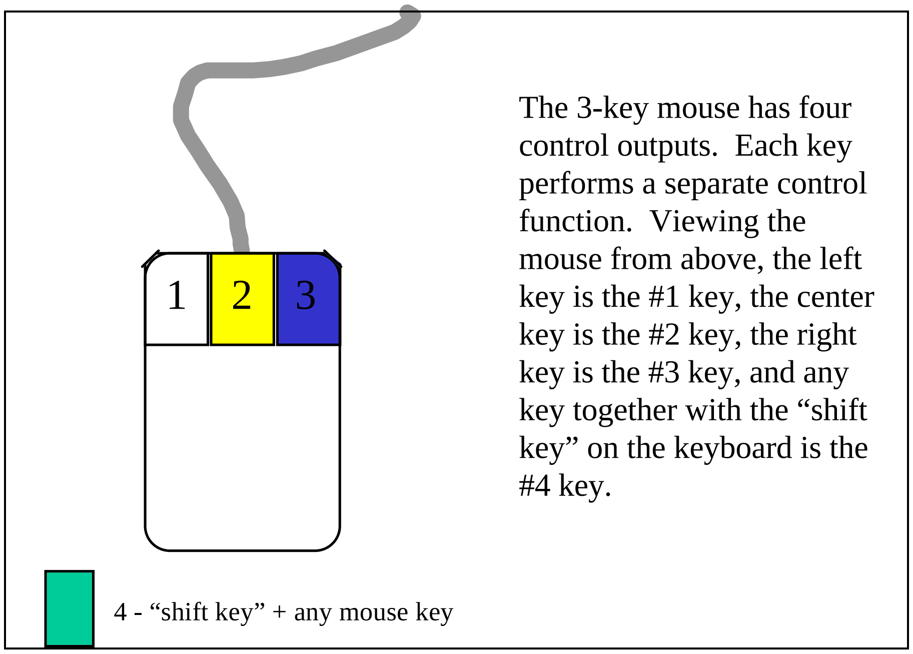
Janus is a high resolution interactive ground combat computer simulation. It allows you to simulate the combat interaction of friendly and enemy forces. Each element is viewed on the Janus screen as an icon. The icon may represent one or more pieces of equipment. For example, one icon may represent one tank, or it may represent a platoon of tanks. This tutorial will show you how to tell the difference.

3. TUTORIAL OVERVIEW

This document is a tutorial for the beginning Janus user on the HP workstation. We provide a step-by-step method to learn the fundamentals of Janus quickly. The first time user should follow the tutorial in sequence. The more experienced user can skip any section that is not relevant. This document does not cover all functions of Janus. We introduce the basic functional commands and features of the model that allow you to use Janus as a research tool.

4. COMMANDS

During the execution of Janus, the primary input method for commands is a three-key mouse. The three-key mouse and the keyboard provide four basic Janus command inputs.



The 3-key mouse has four control outputs. Each key performs a separate control function. Viewing the mouse from above, the left key is the #1 key, the center key is the #2 key, the right key is the #3 key, and any key together with the “shift key” on the keyboard is the #4 key.

The Janus users manual assigns colors to these mouse command functions:

#1 key -	white key
#2 key -	yellow key
#3 key -	blue key
#4 key -	green key

We will use the following commands and terms frequently in this tutorial:

“left click”	-	(the most common control function) press the #1 key.
“center click”	-	press the #2 key.
“right click”	-	press the #3 key.
“shift click”	-	press any key while holding the shift key down.
“double click”	-	press the appropriate key twice rapidly.
“icon”	-	a graphical representation of a piece of equipment. The icon may represent one or more elements. For example, one icon may represent a platoon of tanks or only one tank.
“icon window”	-	a fully reduced window (looks like an icon).

We will often use the preceding control functions together with a reference to a specific icon, command function, or geographic feature. For example: “left click an icon” means to move the mouse on its pad until the screen cursor is positioned on an icon, then press the left mouse key.

5. OTHER COMMANDS

There are two “enter” keys on the keyboard. In Janus, these keys have different functions. We will refer to the enter key located in the main section of the keyboard, usually just above the right shift key, as the “return” key. We will refer to the other enter key, located in the expanded section of the keyboard on the numeric keypad at the far right, as the “enter” key.

6. LOGGING ON

Each HP terminal in the computer lab is associated with a specified number of Janus workstations. Before logging-on, you must know the workstation numbers for your terminal. Make note of the workstation numbers because they will become important later in the tutorial. (Your workstation number is located in the XASSIGN.DAT file in the HP file manager).

Note to first time UNIX users: If at any time while in an “hpterm” or “xterm” window (these windows are described later) and you are unable to make the screen respond to your commands, the screen appears to freeze, or you have generally lost control of the situation, push the “control” key and the “C” key at the same time. This will terminate any program that is currently running and allow you to start over from the beginning.

To log-on: With the mouse, left click on the space provided for your username. The blinking cursor should now appear in the leftmost area of the username space. Type your username in the space provided and press the ‘tab’ key on the keyboard. The blinking cursor should now appear in the space provided for your password. Type your password in the space provided and press the return key.

A U. S. Government warning message should appear with an hourglass ‘icon’ in the center of the screen followed shortly by an HP ‘desktop’ screen. At the extreme bottom of the desktop and just to the left of center, there is a computer console icon (looks like a computer). Left click on this computer icon. A window should appear with the word “hpterm” centered at the top (this is called an hpterm window). At the top of the hpterm window there is a prompt (“>”) followed by a blinking cursor. Type the word “xterm” (no quotation marks) and press the return key. (See below.)

```
hp1>xterm (return)
```

Now a new window will appear labeled “xterm” (this is an xterm window).

After the xterm window opens, you may now minimize the hpterm window using the following procedure:

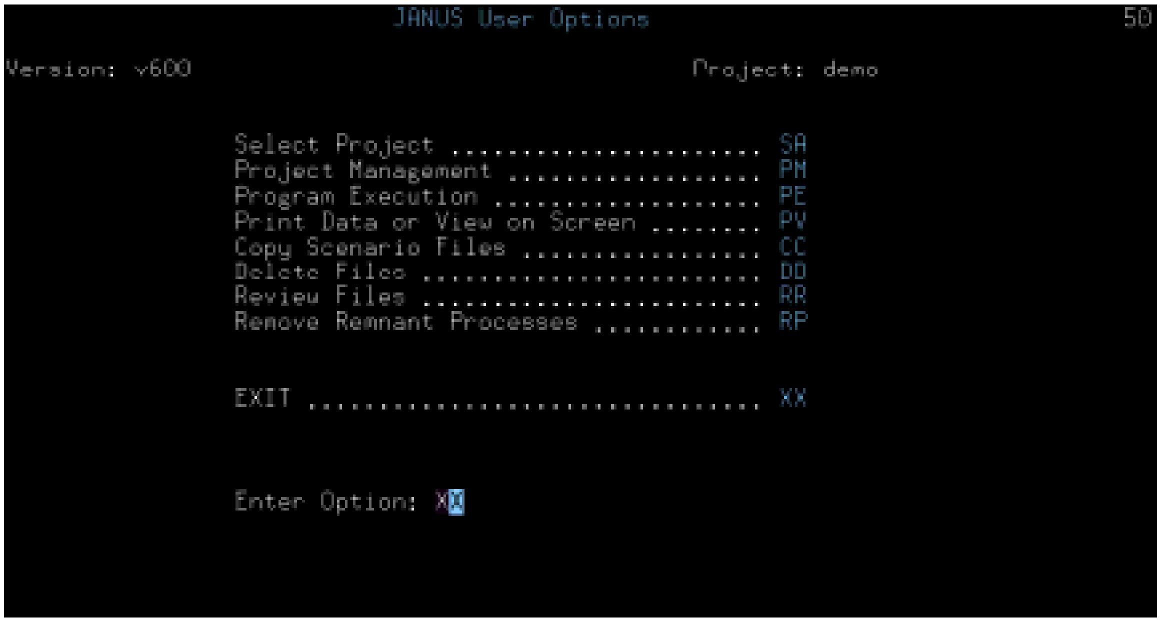
1. Left clicking on the "•" at the upper right corner of the hpterm window (next to the small box in the extreme upper right corner).

2. The hpterm window should reduce to an icon window which is at the upper left corner of your screen (labeled "hpterm" at the bottom).

Return to the xterm window by left clicking anywhere inside of it. Now type "menu" as shown below and press the return key:

`hp1>menu` (return)

This will start Janus and the following Janus User Options window will appear:



```
                         JANUS User Options                          50

Version: v600                                    Project: demo


               Select Project ....................... SA
               Project Management ................... PM
               Program Execution .................... PE
               Print Data or View on Screen ......... PV
               Copy Scenario Files .................. CC
               Delete Files ......................... DD
               Review Files ......................... RR
               Remove Remnant Processes ............. RP


               EXIT ................................. XX



               Enter Option: XX
```

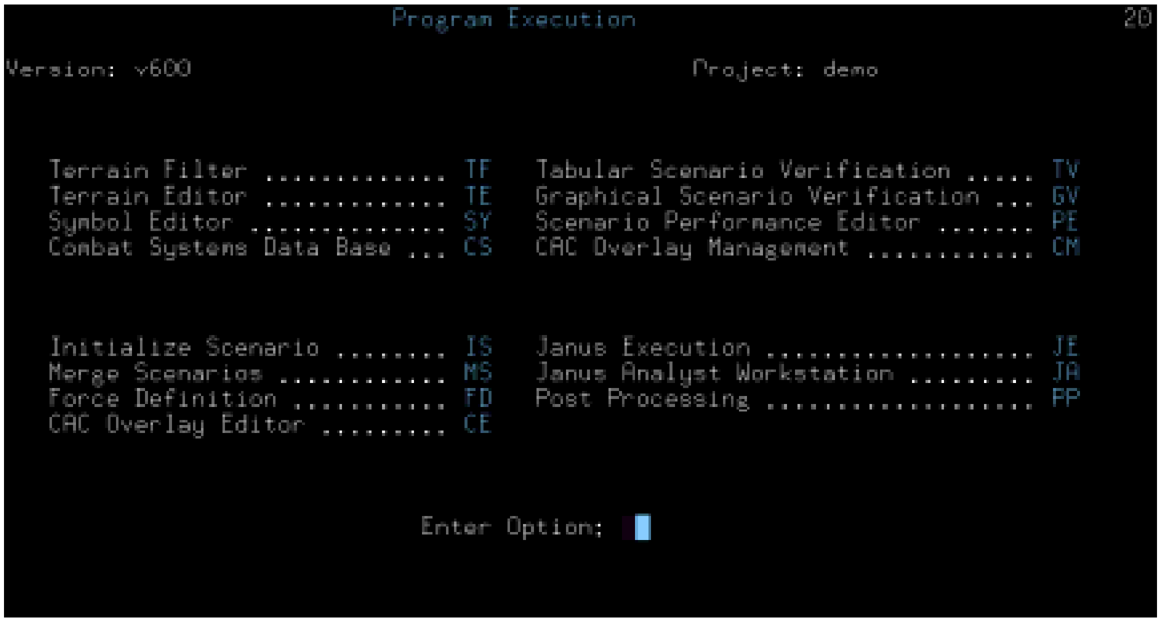
You are now logged-on and ready to proceed in Janus.

7. JANUS ADMINISTRATIVE COMMANDS

This main Janus menu screen lists several command options, two-letter option codes, and a user prompt at the bottom of the screen which reads "Enter Option". For this tutorial, we will only be concerned with the "Program Execution" option which has the two-letter code "PE". Type "PE" at the Enter Option prompt and press the return key as follows:

```
        Enter Option:  PE (return)
```

The following Program Execution screen will appear:



```
                          Program Execution                                     20

Version: v600                                           Project: demo


   Terrain Filter .............. TF    Tabular Scenario Verification ..... TV
   Terrain Editor .............. TE    Graphical Scenario Verification ... GV
   Symbol Editor ............... SY    Scenario Performance Editor ....... PE
   Combat Systems Data Base .... CS    CAC Overlay Management ............ CM


   Initialize Scenario ......... IS    Janus Execution ................... JE
   Merge Scenarios ............. MS    Janus Analyst Workstation ......... JA
   Force Definition ............ FD    Post Processing ................... PP
   CAC Overlay Editor .......... CE


                            Enter Option:
```

This screen allows you to edit and adjust many aspects of the scenario prior to a run. For now we are only interested in the "Janus Execution" option (two-letter code "JE"). Type "JE" at the Enter Option prompt and press the return key as follows:

```
        Enter Option:  JE (return)
```

The following line should appear in the xterm screen that you started earlier:

```
        Enter the Scenario Number for this run (1-999): [     ]
```

Highlight the xterm screen by moving your mouse pointer anywhere on the xterm screen and left click once. This tutorial was designed around scenario 311, an amphibious landing at Camp Pendleton, CA, so this is the scenario we recommend. Enter the number of the scenario that you wish to run inside the square brackets and then push the return as follows:

```
Enter the Scenario Number for this run (1-999): [311] (return)
```

A new line will appear asking for a run number between 1 and 99. The "run number" is an administrative control that allows you to vary your scenario and capture output from different variations in separate output files. For now, the run number you select is not important. We recommend run number 1. Enter the appropriate number in the square brackets and press the return key as follows.

```
Enter the Run Number for this run (1-99): [01] (return)
```

At this point there are two possible outcomes:

1. Two new lines appear which read:

```
RUN number XXX has already been made.

Enter XX to stop, or press RETURN to continue [    ]
```

If this occurs, a previous student has run the scenario using the same run number that you just inputted. Simply press the return key to continue (you will write over the previous student's output file, but that's OK).

2. One new line appears which reads:

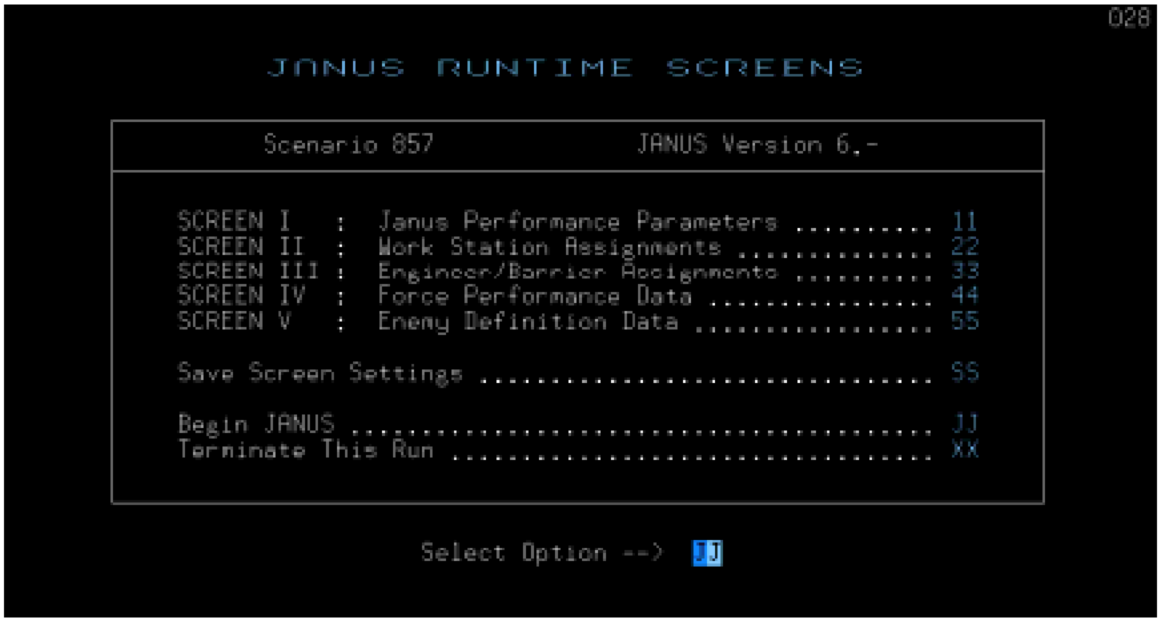
```
Enter XX to stop, or press RETURN to continue [    ]
```

If this occurs, simply press the return key to continue.

Next, a Janus Run Types Menu will appear which prompts you to select a run type. We want a NORMAL Run, so type a "1" at the Select Run Type and press the return key as follows:

```
Select Run Type [ 1 ] (return)
```

Now, the xterm window will display lines indicating that files are being read by the HP system, and a new menu will appear titled "Janus RUNTIME SCREENS". The window looks as follows:



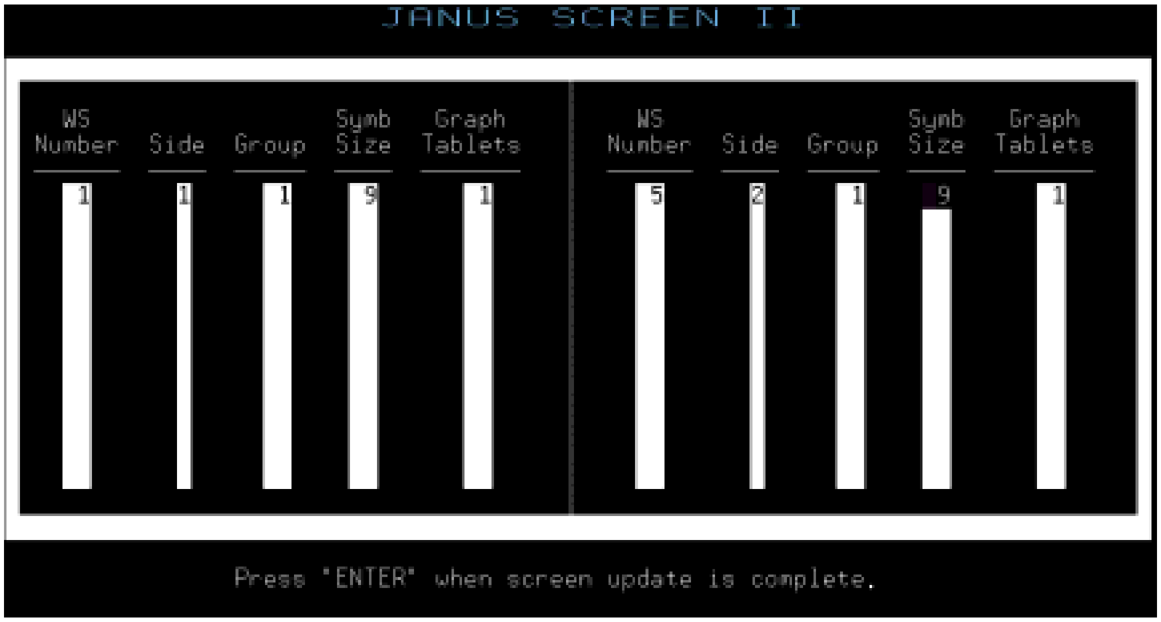
```
                                                         028
              JANUS  RUNTIME  SCREENS

       Scenario 857              JANUS Version 6.-

   SCREEN I   :  Janus Performance Parameters ..........  11
   SCREEN II  :  Work Station Assignments ..............  22
   SCREEN III :  Engineer/Barrier Assignments ..........  33
   SCREEN IV  :  Force Performance Data ................  44
   SCREEN V   :  Enemy Definition Data .................  55

   Save Screen Settings ................................  SS

   Begin JANUS .........................................  JJ
   Terminate This Run ..................................  XX

              Select Option -->  JJ
```

This screen allows you to review information about friendly and enemy forces associated with this specific scenario, allocate Janus workstations, allocate obstacle resources to units, and review some of the parameters Janus uses to operate. Additionally, we will start our Janus run from this screen. For now, we are only concerned with Janus work station assignments, and starting the simulation.

Select SCREEN II : Work Station Assignments option (two letter code "22"), enter the code by typing "22" at the Select Option prompt, and press the enter key (this is where we begin to use the numeric keypad enter key):

`Select Option -- > 22` (enter)

The screen will now change to "Janus SCREEN II". This screen allows the user to set the proper work station numbers so that the friendly and enemy forces in your scenario appear on your monitor. Make your screen look as follows:


JANUS SCREEN II

WS Number	Side	Group	Symb Size	Graph Tablets	WS Number	Side	Group	Symb Size	Graph Tablets
1	1	1	9	1	5	2	1	9	1

Press "ENTER" when screen update is complete.

You must verify that the two work station numbers you were given are listed in the WS Number columns and that these two work station numbers correspond to Side 1 (friendly forces), and Side 2 (enemy forces). For example, if your work station numbers are 1 and 5, then there should be a “1” and a “5” in the WS Number column with either a “1” or a “2” in the Side column for work station 1, and a “1” or a “2” (whichever is **not** associated with work station 1) in the Side column for work station 5. If the workstation numbers do not match yours, change them. (Janus is capable of playing up to six sides. Ensure workstations are properly assigned when playing multiple sides). Press the return key and the blue blinking cursor will cycle through the positions on the screen. If you go too far, you can use the “backspace” key to cycle backwards through the positions (or you can move the mouse pointer over the location of the option you need to change and left click). When the blue blinking cursor is over the number to be corrected, type in the correct number.

NOTE: You cannot assign two “sides” to one work station. You should have at least two work station numbers and each of these numbers should be assigned a different “side”.

The “Group”, “Symb Size”, and “Graph Tablets” positions should be assigned the numbers “1”, “9”, and “1”, respectively (just as you see in the screen representation above). If they are not correct, change them to the appropriate numbers. (See chapter 6, Software User’s Manual, Version 6.3, for descriptions of “Group”, “Symb Size”, and “Graph Tablets”).

When all work station numbers are correct , then press the enter key. This will return you to the Janus RUNTIME SCREENS menu.

8. PREPARATION FOR MAP ENVIRONMENT

You are now ready to enter the map environment. Find the two-letter code for "Begin Janus" (it should be "JJ"), enter the code at the Select Option prompt and press the enter key:

```
Select Option -- >  JJ (enter)
```

At this point you will see in the xterm window that Janus is reading the terrain files and setting up the "Battlefield Graphics windows." You will know that the setup is complete when the following two conditions are met:

a. The text in the xterm window will read:

```
            Scenario XXX Forces:

                  Arty  DF    Total
      Side        Units Units Units
        1           30   147   177
        2           13   165   178
```

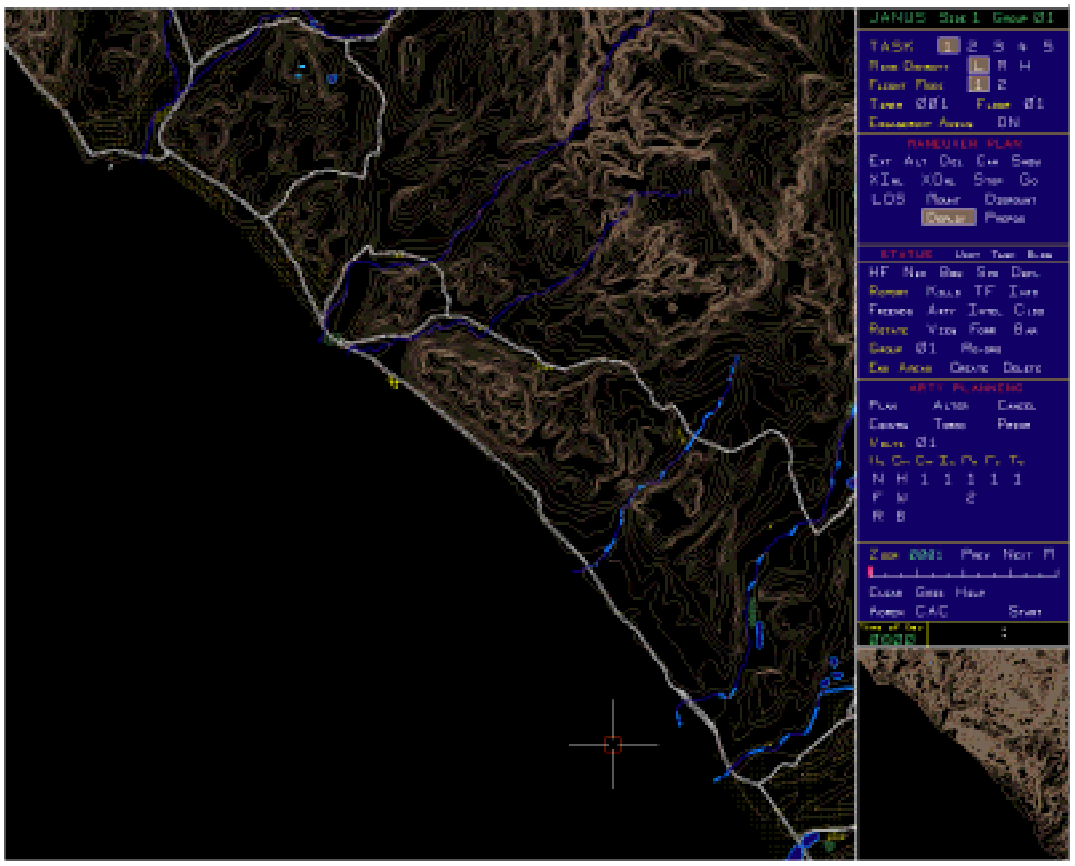
NOTE: Your numbers may not match the above example - they are scenario dependent.

b. An "icon window" will appear for each of the "sides" you set up in the Janus Workstation menu. For example, if you are running a scenario with "side 1" and "side 2", and you set up Janus Screen II (Workstation menu) to display both "sides" on your HP workstation, then there should be two icon windows in the upper left corner of the screen labeled "side 1" and "side 2".

9. HIGH RESOLUTION BATTLEFIELD GRAPHICS

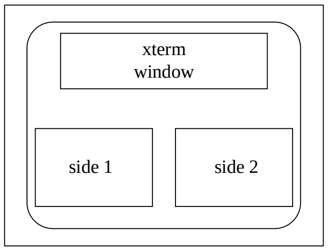
Note: Janus uses digitized terrain developed by the Defanse Mapping Agency (DMA). The terrain appears in a form familiar to military users displaying elevation contours, roads, rivers, vegetation, and urban areas. The colors used in the model are slightly different than used on military topographic maps; elevation contours are brown, rivers and other bodies of water are blue, roads are brown or gray with white outlines, urban areas are yellow and vegetation is green. Other features such as pipelines, power lines, railways and fences can also be portrayed.

Now, double left click on the icon labeled "side 1". The resulting window should look similar to the figure below. The right side of the screen is an interface panel which you will use to control the forces displayed on the map.



You can "resize" the battlefield graphics screen by moving the mouse pointer to the upper right corner of the screen until the pointer changes to a "corner symbol". When the corner symbol appears, left click with the mouse, hold the key down, and move the mouse to the left and down to make the screen smaller. We suggest you shrink the side 1 screen to fit in the lower left corner of the monitor.

Now, double left click on the icon labeled "side 2". The resulting window displays the side 2 initial positions just like the side 1 screen. You can also resize this screen in the same manner that you resized the side 1 screen. We recommend you grasp the upper left hand corner of the side 2 screen and move it to the lower right corner of your monitor workspace. Your computer monitor should now look like this:



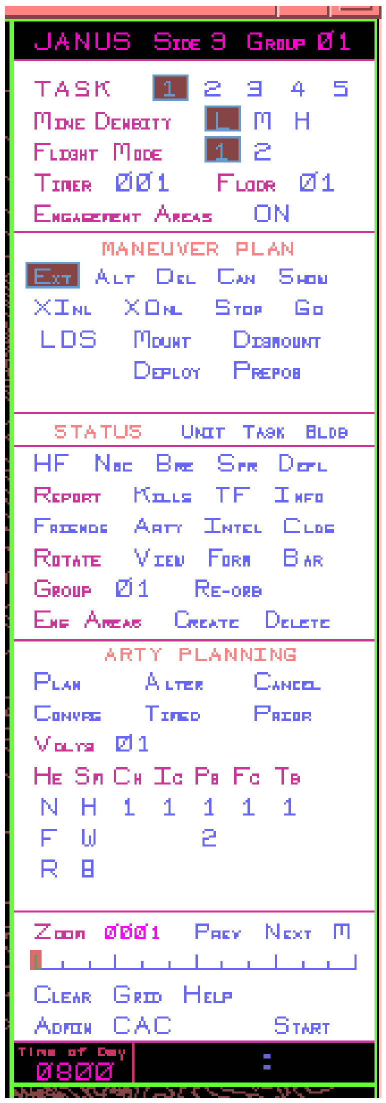
You are now ready to issue initial “orders” to the units on the screen. Further instructions for deployment, assignment of sectors of fire, establishment of routes, etc., will be covered in the next section of this tutorial.

10. SCENARIO PREPARATION COMMANDS

The following section describes the Janus interface panel. It will explain how to input commands to battlefield units. The scenario we are working with (scenario 311) portrays an amphibious assault landing at Camp Pendleton, CA. The landing force (side one) is composed of several battlefield systems, including Amphibious Assault Vehicles and LCACs each carrying an M1 tank. The defending force (side 2) consists of tanks, air defense weapons, and barriers positioned in a valley along the coast of Camp Pendleton. Enlarge the side 2 Battlefield Graphics Window and familiarize yourself with the position of the forces. Note that there are three entities in the upper left hand corner of the screen. There is a T-80 tank, an artillery piece, and a Hind helicopter. You will be experimenting with the Janus Interface Panel commands using these battlefield systems. These instructions will guide you through each command and familiarize you with operating in the Janus environment. You should read through section 1 (administrative commands) of the Interface Panel and then move on to section 2 (maneuver plan) where you will begin to input commands to the units on the battlefield.

At this time, you may want to run the Camp Pendleton scenario so that you can visualize what a complete Janus scenario run looks like. In order to do this, go to paragraph 14 (Start Simulation) on page 23 of this tutorial. Follow the start steps and watch the complete amphibious assault. When all the action is complete (approximately simulation time 75:00) or when you have seen enough action, stop the simulation using the procedure outlined in paragraph 15 (Stop Simulation and Log Off) on page 23. Paragraph 15 also outlines the correct log off procedure you will follow at the conclusion of your Janus session.

11. JANUS INTERFACE PANEL



Before beginning any scenario, the user must take certain preliminary steps. To enable the user to do so, Janus displays options along the right side of the screen in the form of the menu you see here (the Janus Interface Panel). The commands are selected by moving the mouse cursor over the command and clicking the left mouse key. This menu appears at the right of the terrain map for both side 1 and side 2 forces. The menu has the five sections listed below:

a. Administrative (TASK, MINE DENSITY, FLIGHT MODE, TIMER)
b. MANEUVER PLAN
c. STATUS
d. ARTY PLANNING
e. Administrative Screen Control
f. CLOCKS

SECTION 1 - ADMINISTRATIVE

a. **TASK**. The five numbers to the right of TASK are the task force numbers of units on the screen. Pick one of these numbers to display that task force number slightly above and to the right of the unit symbol. Try this command by choosing Task 1 or 2 on either the side 1 or side 2 window. You should see small numbers appear by various icons on the map.

b. **MINE DENSITY**. The three options in this section are: L for low (40 mines), M for medium (80 mines), and H for high (160 mines). They refer to the density in a 50,000m^2 area. Mine density applies to FASCAM and vehicle-emplaced minefields, but not to hand-emplaced minefields.

c. **FLIGHT MODE**. Flight mode 1 causes aircraft to fly low and slow and flight mode 2 causes aircraft to fly high and fast.

d. **TIMER**. The TIMER function is used with other functions to plan future events such as prearranged artillery fire or scheduled unit movements.

e. **ENG AREAS ON**. When enabled, this function activates engagement areas for direct fire weapons. Units will engage other units within engagement areas according to the firing criteria specified previously on Janus Screen IV.

f. **FLOOR**. The FLOOR function sets the current floor number for buildings. This function is used in conjunction with MOUNT, DISMOUNT, and LOS.

SECTION 2 - MANEUVER PLAN

a. **DEPLOY Command**: The Deploy command allows you to reposition your forces prior to the start of the simulation. Use the deploy command now to reposition the Tank, Artillery Piece, and Helicopter that are positioned in the upper left corner of the side 2 graphics display. You can place them wherever you like for now. Later, you will want to deploy them tactically for inclusion in the simulation battle.

1. Left click on the Deploy command.

2. Left click on the icon you want to move (NOTE: the cursor cross disappears and the mouse freely moves the icon about the window).

3. Move the icon to the new position and left click. The cursor cross now returns and the icon is located at the new position.

4. Do not forget to check the element's new field of view (see VIEW command in the following section) to ensure that the weapon system is oriented in the desired direction.

b. **EXT Command**: Routes can be entered prior to the start of the simulation and changed while the simulation is in progress. To assign a route to a unit:

1. Left click the EXT command in the Maneuver Plan function group. This should highlight the command.

2. Left click on the icon whose route you are building. A white triangle should appear over the icon. Wherever you move the cursor, an orange line follows.

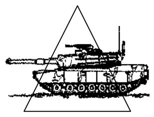
3. Move the orange line to the next checkpoint and left click. A new triangle should appear where you clicked with a white line connecting the two triangles. In this way you can plot routes as a succession of straight line segments.

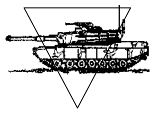
4. At the final checkpoint for the route, center click. This will end the route planning for that unit and the orange line that follows the cursor will be turned off.

5. Refer to the ALT and DEL commands, Sections 2b. and 2c. respectively, to alter or delete a route.

When you are finished setting routes, left click the “CLEAR” command in section 5 of the Interface Panel to clear the screen of all route lines. Try creating routes for each your three battlefield systems on side 2.

NOTE: There are two different types of white triangles which form the corners of your routes.

1. A **GO** triangle looks like the one shown here: 

2. A **STOP** triangle looks like this: 

3. STOP triangles are designed to hold a unit at the stop location for a specific period of time before the unit is allowed to move to the next leg of its route. Timed triangles are beyond the scope of this tutorial.

4. To change a STOP triangle to a GO triangle (or vice versa):

    - Left click on the EXT command.

    - Left click on the icon you wish to change. This displays the icon’s current route.

    - Right click on the triangle you want to change from a STOP to a GO triangle. The triangle should change to a GO triangle. The same procedure works for changing a GO triangle to a STOP triangle.

c. **ALT Command**: Allows you to alter a previously entered route.

1. Left click on the “ALT” Command. This should highlight the command.

2. Left click on the icon whose route you wish to alter. The icon’s current route will be displayed in white.

3. You may now left click on, and move any triangle except the one covering the icon.

4. Place the triangle in the new location and center click.

d. **DEL Command**: Allows you to delete a single leg from a previously entered route.

1. Left click on the "DEL" Command.

2. Left click on the icon whose route you want to alter. The icon's current route will be displayed in white.

3. Left click on the triangle you want to delete and it will disappear leaving that portion of the route that exists without the triangle you deleted.

e. **CAN Command**: If you want to cancel a route:

1. Left click the task force number of the unit you wish to affect.

2. Left click the "CAN" command. The command should be highlighted.

3. Left click on the icon whose route you want to delete. The current route will apper in white. Left click again and the route should be removed. The unit becomes stationary.

f. **SHOW Command**: If you want to view the routes again:

1. Left click the task force number of the unit whose route you want to see. The number should be highlighted and task force numbers will appear over all units on the simulation screen that are part of that task force.

2. Left click on the "SHOW" command. The assigned routes for the task force you've selected will be displayed in orange.

g. **XINL Command**: Allows you to give different icons the same route so that they follow behind one another (not needed for this tutorial).

h. **XONL Command**: Allows you to give different icons the same route so that they move abreast (not needed for this tutorial).

i. **STOP** and **GO Commands**: These commands allow you to stop an entire task force's movement, or start an entire task force's movement.

1. Left click on the task forces you want to effect.

2. Left click on the GO command and the entire task force will initiate movement (any current STOP triangles will change to GO triangles).

3. Left click on the STOP command and the entire task force will stop movement (current GO triangles will change to STOP triangles).

j. **LOS Command**: Displays an element's field of view and what it can see (Line of Sight).

1. Left click on the LOS command. The command should now be highlighted.

2. Left click on the icon for the element you want to check.

3. A sector arc will appear that shows the line of sight in the field of view that was last assigned to that unit (see section 3 - View command). The dashed white line indicates the center of the sector between left and right lateral limits. The orange lines are areas where vision is unimpeded while deadspots (areas the element cannot see) are represented by missing segments in the orange radial lines. The maximum visibile range of the element is represented by the far white arc. The maximum range of the primary weapon system is represented by the purple arc.

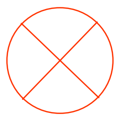
4. To check the LOS for a location with no icon, move the cursor to the new position and click. A full 360 degree field of view will appear. If you find this position's LOS to be more favorable, then deploy (see para l below - Deploy command) the unit to the center of the LOS circle and recheck the element's LOS to verify proper placement.

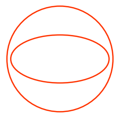
k. **MOUNT Command**: Allows you to mount one system on another for movement and protection (infantrymen to mount in fighting vehicles, tanks to mount in LCACs, etc.). This function will not be used in this tutorial.

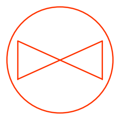
l. **DISMOUNT Command**: Allows you to dismount one system from another.

m. **PREPOS Command**: Allows you to establish prepared positions that will increase your elements' survivability and concealment. Janus will allow you up to 600 prepared positions. The positions are depicted on the graphics screen as follows:

Infantry Prepared Position



Vehicle Prepared Position 

Helicopter Pop-up Position 

1. To create a prepared position, left click on the PREPOS command, move to the location where you want to place the prepared position. Left click to create an infantry prepared position. Center click to create a vehicle prepared position. Right click to create a helicopter pop-up position. If you place a prepared position on top of an icon, the icon is now in a prepared position.

2. To remove a prepared position, left click on the PREPOS command, then shift click on the prepared position you want to remove.

SECTION 3 - STATUS

This section allows you to change several of your units' parameters, review data concerning friendly forces, adjust you units' field of view and other functions.

a. **UNIT/TASK/BLDG**. The first commands under Status are the Unit/Task/Bldg commands. These commands determine the scope of the changes you make in parameters. For example, if you left click on Unit, then left click on NBC, and then left click on one icon, only that icon will be effected. However, if you left click first on Task (task force), then on NBC, and then on an icon, **<u>all</u>** the elements in that icon's task force will be effected. Likewise, if you first left click on Bldg, any adjustments you make to an icon that is inside a building, will also effect all other elements that are inside the building.

b. **HF**. This function changes the hold fire status of a unit. If a unit is in a HF status, that unit will not engage enemy units until HF is disabled. After left clicking the desired TASK number in section 1, UNIT in section 3, and HF, the map will display the HF status of the corresponding units. There will be a white "V" under the icon of units that are in HF status, and a white line under units that are not in HF status. The user can change the HF status of a unit by clicking on the unit icon. You can change an entire task forces' HF status by highlighting TASK (instead of UNIT) prior to clicking on the unit icon.

c. **NBC**. If NBC is enabled ("V" displayed), the unit is in protective gear. NBC status is enabled/disabled similar to HF.

d. **BRE**. If BRE is enabled (“V” displayed), those units with breach capability will begin conducting breaching operations, even if no minefield/obstacle is in the vicinity of the enabled unit. Breaching units move slower than non-breaching units. BRE status is enabled/disabled similar to HF.

e. **SPR**. If SPR is enabled (“V” displayed), those units are in a sprint mode and will move at maximum possible speed. SPR does not affect flying units. SPR status is enabled/disabled similar to HF.

f. **DEFL**. If DEFL is enabled (“V” displayed), those units are in full defilade. When in full defilade, the unit cannot engage enemy units except in self defense, although they can see the enemy. Units in full defilade are not visible until the enemy is within 50 meters. Units in partial defilade (white line displayed) will engage the enemy. DEFL status is enabled/disabled similar to HF.

g. **REPORT KILLS**. Selecting KILLS displays original number of friendly systems by type, remaining systems, system losses, and total losses. Selecting KILLS with the center mouse key gives information for selected task force. Selecting KILLS with the left key gives the same information for all forces assigned to the workstation. Selecting KILLS with the right key gives the same information for the entire side.

h. **REPORT TF**. Selecting TF with the center mouse key displays a text report on the currently selected task force. Selecting TF with any other mouse key displays the unit number of each unit in the selected task force on the map below the icon.

i. **REPORT INFO**. Selecting INFO and selecting a specific icon with the center mouse key, displays the same text report as TF for the selected icon. Selecting an icon with the left key displays the unit’s grid location. Selecting an icon with the right key displays the unit’s routes. Selecting an icon with the shift click key displays the unit’s ammunition status.

j. **FRIENDS**. When the user plays fratricide, Janus disables FRIENDS. When the user does not play fratricide, the FRIENDS report displays all same-side units assigned to other workstations and all different side units that are not enemies, if line of sight exists between them and a unit of your workstation.

k. **ARTY**. Selecting ARTY with the left, center, or right mouse key displays currently planned artillery missions. Selecting ARTY using shift click displays all target reference points.

l. **INTEL**. Selecting INTEL with the center mouse key draws a line between the detecting units and all units they currently detect. Selecting INTEL with the left key displays the last 25 units detected within the last ten minutes which are no longer detected. Selecting INTEL with the right key displays aggregation level and formation of each currently detected unit.

m. **CLDS**. Janus can simulate smoke pots. The user inputs smoke into the scenario using JANUS SCREEN III. The smoke pot symbols are a square with a smoke insignia inside. Smoke pots are deployed during the preparation phase of the scenario using the DEPLOY command. The smoke pots must be activated after you start the simulation by clicking the SETOBS command and then clicking on the smoke pot symbol. After clicking on the smoke pot symbol, the symbol will disappear. The CLDS command allows the user to turn the smoke display on and off. When you activate CLDS, all smoke clouds appear on the screen in the form of a circle. When CLDS is off, no smoke symbols appear on the screen, but smoke is still in effect.

n. **ROTATE VIEW/FORM**. These functions allow the user to orient unit view fans and unit formations. Select the desired function, then select the unit icon to be adjusted. Adjust the view or formation by positioning the cursor in the desired direction on the map and clicking the center mouse key.

o. **BAR.** Janus can simulate three different barriers/obstacles. They are road craters, abatis, and minefields. The user inputs barriers/obstacles into the scenario using JANUS SCREEN III. Minefields are represented by rectangular with dimensions as set in the database. Road craters are represented by two diagonal slashs inside a circle. Abatis are represented by an upside down “V” with an attached line. Barriers/obstacles are deployed during the preparation phase of the scenario using the DEPLOY command. Prior to deploying the barriers/obstacles, the user should set their orientation. To set orientation, first click the BAR command. Notice the small white circle in the bottom right corner of the window. The circle represents the barrier’s radius and the white line represents the barrier’s orientation. Moving the cursor to any point on the circle and clicking on the circle changes the barrier’s orientation. Now deploy the barrier/obstacle by clicking DEPLOY and moving the cursor to the barrier/obstacle you wish to deploy. Click that barrier/obstacle and move the cursor to the desired location on the map and click again. The barrier is then deployed with the previously set orientation. Both abatis and road craters must be activated after you start the simulation by clicking the SETOBS command and then clicking on the abatis or road crater symbol. Notice that both abatis and road crater turn from white to purple after activation.

p. **GROUP RE-ORG**. This function allows the user to change a unit’s task force number. Select the desire TASK number, select RE-ORG, then select the desired unit icon. Janus will display the new task force number next to the selected icon. RE-ORG may also be used to transfer units from one workstation to another.

q. **ENG AREAS CREATE**. To create an engagement area, select CREATE and left click on the desired map locations to define the area. Middle click on the final point.

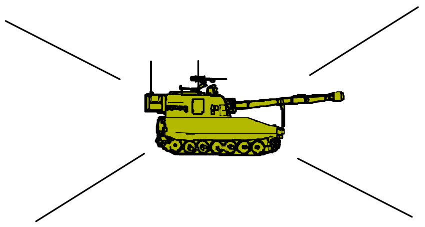
r. **ENG AREAS DELETE**. To delete an engagement area, select DELETE then select the desired engagement area.

SECTION 4 - ARTILLERY PLANNING

This section describes the basics of how to plan artillery. We will ignore most of the commands under artillery planning since they are beyond the scope of this tutorial. For this simulation, you will fire only HE, no timed missions, no priority targets, and no converging sheafs. You will use the Janus Battlefield Graphics window along with the Artillery Planning section of the interface panel. Experiment with these commands using your artillery piece. **Note**: In order for your artillery piece to receive and fire missions, it must be at a stop triangle (see maneuver plan section and ensure your artillery piece is at a stop triangle before proceeding).

a. **PLAN Command**: This command allows you to set-up fire missions.

1. Left click on the PLAN command. It should now be highlighted and artillery capable elements on the battlefield are highlighted with a white X as below:



2. Left click on one of the artillery capable icons and two concentric circles will appear with the icon at the center. The large circle represents the maximum range of the artillery piece for high explosive munitions. The small circle represents the danger close distance (an artillery piece will not fire within this small circle).

3. Move the cursor away from the artillery piece. Note that an orange line connects the cursor to the artillery icon. Choose a target that is outside of the small (danger close circle), but inside of the large (maximum range) circle. Left click on the target. The orange line now

disappears and a white dashed line appears in its place. Additionally, there is now a target reference point at the target location with a fire mission number next to it. The artillery element you picked will fire this mission first when the simulation is started, followed in sequence by any additional fire missions you add.

b. **CANCEL Command**: This command allows you to cancel a previously planned fire mission.

   1. Left click on the CANCEL command.

   2. Left click on the icon whose fire mission you want to cancel. The white circles will appear along with the lines and target reference points which represent that artillery piece's fire missions.

   3. Identify the mission you want to cancel and left click on the target reference point which represents that mission. The mission should disappear (it is now cancelled).

c. To exit the artillery planning mode, left click on any of the commands in the MANEUVER PLAN command group (section 2).

SECTION 5 - ADMINISTRATIVE CONTROLS

a. **ZOOM**. This function allows you to magnify portions of the map. Select the desired magnification by clicking on the zoom scale then click on the map location to be magnified. The counter next to ZOOM indicates the zoom scale.

b. **PREV/NEXT**. These functions allow you to easily switch between the previous (PREV) screen and the original (NEXT) screen.

c. **M**. This function corresponds to the standard military map scale of 1:50,000. Select M, then click on the desired map location. The resulting screen will be in the standard military map scale.

d. **CLEAR**. This function removes temporary text, displays, and graphics from the screen.

e. **GRID**. This function superimposes a UTM grid system on the screen.

f. **HELP**. When this function is selected before another menu option, Janus will display a brief explanation of that option.

g. **ADMIN**. Right clicking on the ADMIN function displays an administrative menu. Menu options include saving the prepared plan, increasing the scenario speed, or ending the scenario.

h. **CAC**. This function allows the user to draw Command and Control graphics on the screen.

i. **START**. When this function is selected, the Planning Phase ends and the Execution Phase begins. All workstations must pick START before the Execution Phase may begin.

SECTION 6 - CLOCKS. This section displays the simulated time of day in 24-hour form, and the elapsed time in minutes and seconds from the start of the simulation.

12. SCENARIO PLANNING

You are now ready to input your plan to the units on the screen. This section of the tutorial includes basic scenario planning. Scenario topics covered include pre-deployment locations, routes, prepared positions, barrier emplacement, artillery fire missions, and close air support. (This tutorial assumes that the user has already created the scenario forces in the force development editor. See chapter 3, Software User's Manual, Version 6.3, for force development).

a. Pre-deployment Locations. When the user develops a new force, the unit icons are initially placed in a line at the upper right or left corners of the map (you saw this earlier with your three battlefield systems). Janus places the new force icons there so the user can easily find and move them to the desired start/pre-deployment locations. Move the units using the previously described DEPLOY command on the interface panel (along the right side of the screen) to tactically appropriate positions given your knowledge of the upcoming battle. Repeat this process until all pre-deployment locations are set. Ensure each unit icon is oriented in the proper direction using the LOS and VIEW commands.

b. Routes. Assign routes to unit icons using the EXT command on the interface panel.

c. Prepared Positions. When preparing a defensive plan, place desired units in prepared positions utilizing the PREPOS command. Ensure these units have proper line of sight using the LOS and VIEW commands.

d. Barrier Emplacement. Deploy barriers similar to deploying units using the DEPLOY command. Ensure that the barriers are oriented in the proper direction using the BAR command. Note that abatis and craters are not obstacles until set after the simulation begins.

e. Artillery Fire Missions. Use the ARTY PLAN command to plan all artillery fire missions. This tutorial covers only HE, immediate fire missions. (See chapter 7, Software User's Manual, Version 6.3, for Artillery Planning involving converged sheafs, timed targets, and priority targets).

f. Close Air Support. Select desired FLIGHT MODE command. Select desired aircraft icon and provide a suitable route using the EXT command. (Helicopters can be programmed to pop up at points along a route by using the POP command after the simulation begins. See chapter 9, Software User's Manual, Version 6.3, for POP command instructions).

13. PLAN SAVE

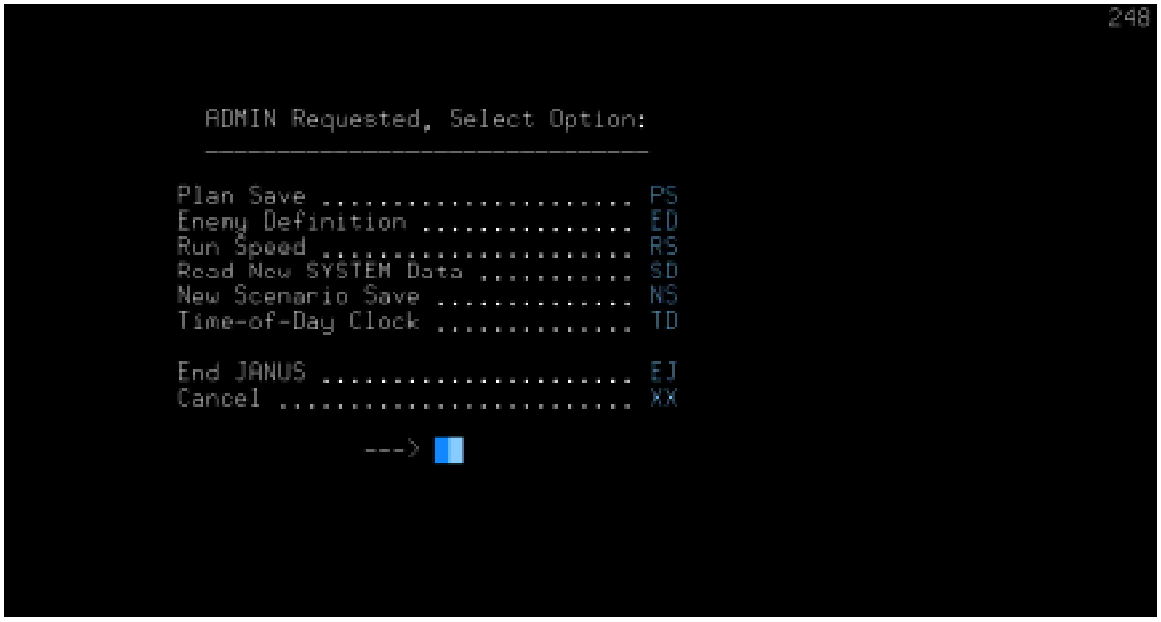
When you have accomplished all the scenario planning, you may want to save the scenario positions. Saving the prepared scenario is important if you want to play the same scenario repeatedly. To save the scenario, click the ADMIN command, then select Plan Save (PS) or NEW SCENARIO SAVE (NS) from the ADMIN menu. Select desired save option and return. Janus will return to the map window.

14. START SIMULATION

To start the simulation, select the START command on the interface panel for all "sides". The xterm window will appear. The screen will prompt you to type "RR" when ready. Type "RR" then return. The next prompt is whether you want to perform a PLAN SAVE now. (This is another method to save the scenario). Type "Y" then return to save the plan. The screen will then ask for the checkpoint frequency. (Frequency "3" is recommended. If you are not saving the run, the frequency can be left blank.). Type frequency then return. This completes the START process and the simulation begins. You can now experiment with the Janus interface panel commands during the simulation run.

15. STOP SIMULATION AND LOG OFF

After the simulation run has begun, you may stop the simulation and exit at any time. Stop and exit the simulation by right clicking on the ADMIN command in section 5 of the Janus interface panel. This will bring up a window that looks as follows:



```
                                                        248

        ADMIN Requested, Select Option:
        _______________________________

      Plan Save ...................... PS
      Enemy Definition ............... ED
      Run Speed ...................... RS
      Read New SYSTEM Data ........... SD
      New Scenario Save .............. NS
      Time-of-Day Clock .............. TD

      End JANUS ...................... EJ
      Cancel ......................... XX

                  --->
```

To end the simulation, enter the two letter code "EJ" at the prompt at the bottom of the ADMIN window, and press the enter key. This will end the simulation run and bring you back to the Program Execution screen. Place the mouse pointer anywhere on this Program Execution screen and right click. Now, you are returned to the Janus User Options menu. Enter "XX" at the Enter Option prompt at the bottom of the screen and press return or enter. You have now exited Janus and you should only see the HP desktop with an icon bar across the bottom of the screen. In the extreme lower right corner of the icon bar is an icon key titled "EXIT". Left click on the "EXIT" key and a "Logout Confirmation" window will appear in the middle of the computer screen. Left click on "OK" and the system will automatically log you off and return to "prepared to log-on" status. You have now successfully completed this tutorial.

16. CONCLUSIONS

This tutorial is an adequate introduction to the Janus simulation. It will accustom you to Janus and allow you to conduct a simple combat scenario. As you become more familiar with Janus, you may want to explore the more complicated and realistic aspects of the system.

LIST OF REFERENCES

1. Janus Resource Data Page from Army Modeling and Simulation Resource Repository, (http://hp01.arc.irquest.com/mosaic/185.html).

2. *Janus Version 6.3 User's Manual, Simulation*, Training & Instrumentation Command, Orlando, Florida.

3. Simulation Interoperability Standards Organization (SISO) page describing liaison with IEEE, (http://siso.sc.ist.ucf.edu/stdsdev/index.htm).

4. Miller, Duncan C., *Mapping HLA to DIS*, Briefing Slides, 11 December 1995.

5. Simulation Interoperability Standards Organization, *DIS++ / HLA Frequently Asked Questions,* (http://siwg.dra.hmg.gb/HLA/mirror/www.dmso.mil/projects/hla/faq/).

6. E-mail from Dr. Donald Ponikvar, former Chief of Staff , HLA Technical Support Team, SUBJECT: HLA Background, 7 December 1996.

7. Memorandum from Under Secretary of Defense, For Secretary of the Army, SUBJECT: DoD High Level Architecture (HLA) for Simulations, 10 September 1996.

8. Defense Modeling and Simulation Office, *High Level Architecture Object Model Template,* Version 1.0, DMSO, 15 August 1996.

9. Defense Modeling and Simulation Office, *High Level Architecture OMT Extensions*, Version 1.0, DMSO, 20 August 1996.

10. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W., *Object-Oriented Modeling and Design*, Prentice-Hall, Inc., 1991.

11. Lutz, Robert, *HLA Object Model Development: A Process View*, Spring Simulation Interoperability Workshop, (http://siso.sc.ist.ucf.edu/siw/97spring/papers.htm), March 1997.

12. Jackson, Leroy A., Wood, J. Ralph, *Exploiting the High Level Architecture for Analysis in Advanced Distributed Simulation*, Spring Simulation Interoperability Workshop, (http://siso.sc.ist.ucf.edu/siw/97spring/papers.htm), March 1997.

13. Parish, Randall M. and Alvin D. Kellner, *Target Acquisition in Janus Army*, U.S. Army TRADOC Analysis Command, White Sands Missile Range, NM, October 1992.

14. Kellner, Alvin D., MEMORANDUM FOR RECORD, SUBJECT: NVEOD Detection in Janus, 14 July 1992.

15. Hoock, Donald, E-mail to CPT Larimer, SUBJECT: Search and Detection Algorithm, 6 December 1996.

16. AEgis Research Corporation, Object Model Development Tool web page, (http://www.aegisrc.com/products/omdt).

BIBLIOGRAPHY

Larimer, Larry R., Buss, Arnold H., Jackson, Leroy A., *Building a Simulation Object Model of a Legacy Simulation*, Spring Simulation Interoperability Workshop, http://siso.sc.ist.ucf.edu/siw/97spring/papers.htm, March 1997.

Loper, Margaret L., *Test Procedures for High Level Architecture Object Model Template,* Version 1.0, 5 September 1996.

Defense Modeling and Simulation Office, *High Level Architecture Interface Specification*, Version 1.0, DMSO, 15 August 1996.

Defense Modeling and Simulation Office, *HLA Rules*, Version 1.0, DMSO, 15 August 1996.

Calvin, James O. and Richard Weatherly*, An Introduction to the High Level Architecture (HLA) Runtime Infrastructure (RTI)*, undated.

Janus Version 6 Data Base Manager's Manual, Training & Instrumentation Command, Orlando, Florida, 1995.

Lutz, Robert R., Abstract, *HLA Object Model Template (OMT) Status,* , Johns Hopkins University/Applied Physics Laboratory.

The Janus 3.X/VMS Model Software Design Manual, Prepared for: Headquarters TRADOC Analysis Center, Ft Leavenworth, KS, Prepared by: Titan, Inc. Applications Group, Leavenworth, KS, 1993.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center..2
   8725 John J. Kingman Road., Ste 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library..2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Professor Arnold H. Buss, Code OR/Bu...2
   Operations Research Dept.
   Naval Postgraduate School
   Monterey, California 93943-5101

4. Director...1
   U. S. Army TRADOC Analysis Center-Monterey
   Monterey, California 93943-5101

5. Major Leroy Jackson..2
   U.S. Army TRADOC Analysis Center-Monterey
   Monterey, California 93943-5101

6. Captain Larry R. Larimer...1
   1612 Larch Street
   Sandpoint, Idaho 83864

7. Chief...2
   Wargame Development Division
   U.S. Army TRADOC Analysis Center-WSMR
   White Sands Missile Range
   White Sands, NM 88002-5502

8. Director...1
   Brigade Models and Simulation Directorate
   U.S. Army TRADOC Analysis Center-WSMR
   White Sands Missile Range
   White Sands, NM 88002-5502

9. Mr. Robert Lutz...1
   Johns Hopkins University Applied Physics Laboratory
   Johns Hopkins Road
   Laurel, MD 20723-6099

10. Division Chief..1
    VIC Division, Production Analysis Directorate
    Operations Analysis Center
    U.S. Army TRADOC Analysis Center-LVN
    Fort Leavenworth, KS 66027